CHƯƠNG I. CÁC NGUYÊN LÝ CƠ BẢN CỦA HỆ ĐIỀU HÀNH

1.1 Sự tiến hoá của hệ điều hành hiện đại

a. Khái niệm hệ điều hành

Hệ điều hành (Operating System - OS, dưới đây viết tắt tiếng Việt là HĐH) là một hệ thống các chương trình (và dữ liệu - *tham số hệ thống*) được cài đặt sẵn (dưới dạng các file trên đĩa từ - băng từ) thực hiện hai chức năng cơ bản:

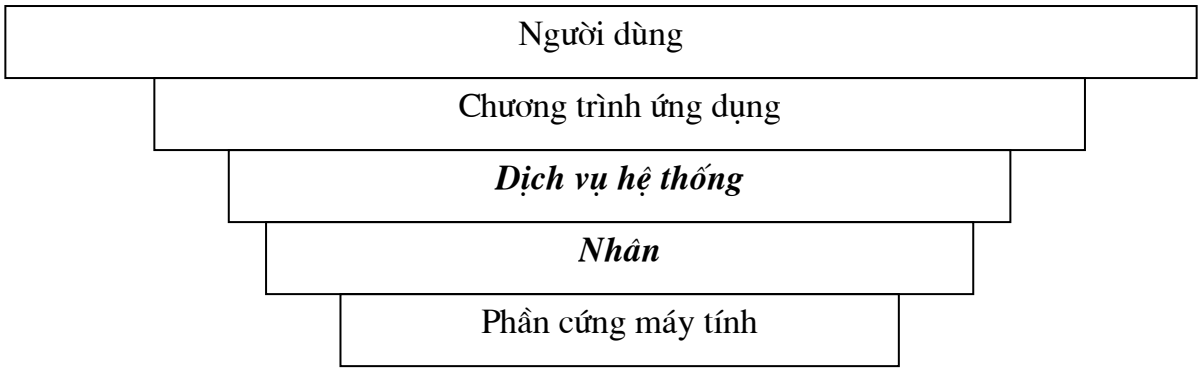
- Chức năng của một hệ thống quản trị tài nguyên: Quản trị, phân phối công việc cho hệ thống thiết bị để hệ thống thiết bị hoạt động hiệu quả nhất,

- Chức năng của một máy tính mở rộng (máy tính ảo): Phục vụ nhu cầu đa dạng của người dùng một cách tốt nhất.

Theo cách nói cụ thể hơn, HĐH là một bộ các môđun phần mềm hệ thống đóng vai trò giao diện giữa chương trình ứng dụng với phần cứng hệ thống, với mục tiêu đạt tới một hệ thống máy tính hiệu quả, tin cậy và dễ sử dụng. Một cách đại thể, tồn tại các chức năng riêng biệt của HĐH như lập lịch làm việc của bộ xử lý (hoặc các bộ xử lý), phối hợp thực hiện các quá trình (QT: process) tương tác nhau, quản lý các tài nguyên hệ thống (chẳng hạn như các thiết bị vào/ra, bộ nhớ trong, File...) ... nhằm *nâng cao năng lực điều khiển và bảo vệ, duy trì tính toàn vẹn hệ thống, thi hành khôi phục lỗi* và *cung cấp một giao diện người dùng*. HĐH thường cấu trúc hai yêu cầu này thành hai lớp: dịch vụ hệ thống và nhân của HĐH.

Dịch vụ hệ thống là những chức năng mức cao được chương trình ứng dụng nhận biết còn *nhân* (thường trực trong bộ nhớ trong) chỉ đảm bảo những chức năng mang tính cơ bản nhất và phụ thuộc vào kiến trúc hạ tầng.

Hình 1.1. mô tả khung nhìn đơn giản về hệ thống máy tính theo cấu trúc lớp. Vị trí của dịch vụ hệ thống trong hình cho thấy vai trò quan trọng của lớp này.



Hình 1.1. Cấu trúc lớp của hệ thống máy tính

• Với ý nghĩa đóng *vai trò như một máy tính ảo*, theo cách nhìn của *người dùng* (từ lớp chương trình ứng dụng), HĐH là sự trừu tượng hóa của hệ thống máy tính được trình diễn bằng các dịch vụ hệ thống: HĐH được chỉ dẫn như là một máy mở rộng (máy tính ảo). Mục đích của lớp dịch vụ hệ thống là nhằm che đậy đi những chi tiết của hệ thống (phần cứng và phần mềm) đối với người dùng.

• Theo cách nhìn của *người quản trị hệ thống*, dịch vụ hệ thống và nhân được coi là người quản lý tài nguyên. Quản lý hệ thống tài nguyên (CPU, bộ nhớ, hệ thống vào ra, file) không chỉ kiểm soát được tình trạng của các tài nguyên mà còn nhằm khai thác

hiệu quả nhất. Một số bài toán điển hình như điều khiển bộ nhớ, lập lịch QT, điều khiển liên QT, điều khiển file, điều khiển vào ra ...

Máy tính mở rộng và quản lý tài nguyên là hai thuật ngữ chung nhất được dùng để xác định một HĐH. Máy tính mở rộng (trừu tượng máy) là mục tiêu thiết kế nguyên thủy đối với HĐH và quản lý tài nguyên giải nghĩa cho việc thực hiện mục tiêu đó.

Thiết kế HĐH truyền thống thường bắt đầu từ yếu tố quan trọng hơn là quản lý tài nguyên, trong khi đó thiết kế HĐH hiện đại lại tập trung nhiều hơn vào yếu tố trừu tượng máy. Và một lẽ tất nhiên là yếu tố nào là quan trọng hơn lại phụ thuộc vào sự quan tâm từ phía người dùng.

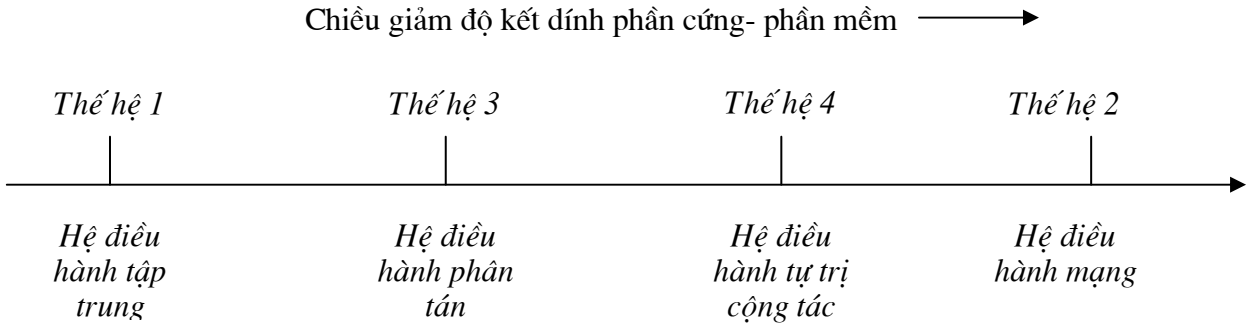
b. Sơ bộ về sự tiến hóa của hệ điều hành

Trong máy tính thuộc các thế hệ đầu tiên không có HĐH. Các thao tác chọn công việc, phân công công việc đều do thao tác viên (và thậm chí ngay chính người lập trình) thực hiện. Theo thời gian, năng lực của máy tính được nâng cao: về tốc độ xử lý của CPU, về dung lượng bộ nhớ, về hệ thống thiết bị ngoại vi, về phần mềm hệ thống cũng như số lượng và năng lực người sử dụng tăng trưởng và vì vậy cần có một hệ thống chương trình điều khiển tự động hệ thống máy tính. Những yếu tố thực tế như vậy làm nảy sinh những điều kiện cần thiết cho việc xuất hiện các HĐH đơn giản.

Lịch sử tiến hóa của HĐH trình diễn một quá trình chuyển hóa từng bước trong việc thiết kế, từ nhấn mạnh chức năng quản trị tài nguyên sang nhấn mạnh chức năng máy tính mở rộng. Theo mô hình trong hình 1.1. thì điều đó được thể hiện việc chuyển hóa từ *nhấn mạnh nhân* sang *nhấn mạnh các dịch vụ hệ thống*.

Theo lịch sử tiến hóa, HĐH hiện đại được phân ra thành 4 thế hệ: HĐH truyền thống (tập trung), hệ điều mạng, HĐH phân tán và hệ tự trị cộng tác. Thế hệ gần đây nhất (hệ tự trị cộng tác) chú trọng thiết kế các ứng dụng phân tán trong môi trường hệ thống mở (bao gồm các thành phần hệ thống hỗn tạp được tích hợp mềm dẻo và có tính khả chuyển nhằm hỗ trợ việc cộng tác thực hiện theo quy mô lớn ở mức ứng dụng).

Dưới đây mô tả sơ bộ về cách thức phân biệt các HĐH này theo (1) độ kết dính phần cứng-phần mềm và (2) tổ hợp mục tiêu-đặc trưng.



Hình 1.2. Phân bố của các thế hệ hệ điều hành theo độ kết dính

• Độ kết dính phần cứng-phần mềm cho biết hệ thống là "tập trung đến mức độ nào", được đo bằng tổ hợp kết dính phần cứng và kết dính phần mềm. Theo đó, phân bố các thế hệ HĐH được sắp xếp như hình 1.2. Tỷ số giữa tổng phí truyền thông liên bộ xử lý so với thời gian truyền thông nột tại bộ xử lý càng thấp thì *kết dính phần cứng* càng chặt. *Kết dính phần mềm chặt* nếu phần mềm điều khiển tập trung và sử dụng thông tin toàn cục.

- HĐH tập trung kết dính phần cứng - phần mềm chặt.

- HĐH phân tán (DOS): phần mềm kết dính chặt trên nền phần cứng kết dính lỏng,

- HĐH mạng (NOS): cả phần mềm lẫn phần cứng đều kết dính lỏng,

- Hệ tự trị cộng tác (CAS) làm giảm kết dính chặt phần mềm (cách nhìn lôgic tập trung của DOS). CAS nằm giữa NOS và DOS.

• Phân biệt HĐH theo tổ hợp mục tiêu-đặc trưng

Bảng 1.1 trình bày sự phân biệt các thế hệ HĐH theo tổ hợp mục tiêu-đặc trưng.

Bảng 1.1. Phân biệt hệ điều hành theo mục tiêu-đặc trưng

Thế hệ	*Hệ thống*	*Đặc trưng*	*Mục tiêu*
1	HĐH tập trung	Quản trị quá trình Quản trị bộ nhớ Quản trị vào-ra Quản trị file	Quản trị tài nguyên Máy tính mở rộng (ảo)
2	HĐH mạng	Truy nhập từ xa Trao đổi thông tin Duyệt mạng	Chia xẻ tài nguyên (liên thao tác)
3	HĐH phân tán	Khung cảnh toàn cục của: Hệ thống file, Không gian tên, Thời gian, an toàn, Năng lực tính toán	Cách nhìn của một máy tính duy nhất của một hệ thống phức hợp các máy tính (tính trong suốt)
4	Hệ tự trị cộng tác	Các ứng dụng phân tán là mở và cộng tác	Làm việc cộng tác (tự trị)

Mục tiêu nguyên thủy của HĐH là *máy tính ảo* (virtual computer). Ba mục tiêu bổ sung là liên thao tác, trong suốt và tự trị hiện vẫn đang là những nội dung nghiên cứu, phát triển.

- Mục tiêu liên thao tác hướng tới năng lực tạo ra điều kiện thuận tiện cho việc trao đổi thông tin giữa các thành phần hỗn tạp trong hệ thống. Đây là mục tiêu gợi mở nguyên thuỷ dẫn tới việc thiết kế HĐH mạng trong một môi trường hỗn tạp.

- Khái niệm *trong suốt* (transparency) và khái niệm *ảo* tương tự nhau ở chỗ cung cấp tính trừu tượng cao cho hệ thống. Điều khác biệt giữa hai khái niệm này là theo tính *ảo*, người dùng có thể nhìn thấy cái họ muốn, trong khi đó tính trong suốt đảm bảo rằng người dùng không nhìn thấy những cái họ không muốn. *Ảo* là mục tiêu quan trọng của HĐH tập trung còn *trong suốt* là mục tiêu quan trọng của DOS. Khái niệm trong suốt cho phép mô tả DOS như một hệ thống cung cấp một khung cảnh lôgic của hệ thống cho người dùng, độc lập với hạ tầng vật lý. Người dùng có được cách nhìn của máy tính đơn cho một hệ thống máy tính phức hợp: sự tồn tại của hạ tầng mạng và hoạt động của hệ thống là trong suốt với người dùng. Từ "trong suốt" ở đây được hiểu theo nghĩa 'thuần khiết" của một môi trường thuần nhất.

- Trong suốt là một mục tiêu quá cao. Hơn nữa, không bắt buộc phải luôn cố đạt tới tính trong suốt vì nó bao gói một độ tập trung nào đó. Điều chắc chắn phù hợp với người dùng là họ thích có được cái nhìn riêng về hệ thống. Người dùng cần một môi trường mở không đòi hỏi nhất thiết về tính trong suốt mà chỉ cần hệ thống cung cấp

tính mở để người dùng biến đổi, chuyển, di trú, mở rộng phần mềm ứng dụng của họ một cách độc lập đối với sự hỗn tạp của hệ thống. Lý do là, như lẽ rất tự nhiên, người dùng biết được sự tồn tại của tài nguyên phức và sự hiện diện của các người dùng khác, và người dùng trở thành cộng tác hoàn toàn với hệ thống. Từ đó, hệ thống phần mềm được xây dựng nhờ việc tích hợp các dịch vụ cộng tác, được cung cấp từ các đơn vị tự trị. Kiểu hoạt động như vậy của hệ tự trị cộng tác rất giống xã hội loài người. Hiện tại một số hệ thống phần mềm lớp giữa (middleware) được xây dựng như những phiên bản (version) ban đầu của hệ tự trị cộng tác.

Liên thao tác, trong suốt, và tự trị là những tính chất rất đáng mong muốn. Người dùng không phải (thường là không cần thiết) biết HĐH hiện tại có phải là mạng, DOS, CAS hay không. Hầu hết các HĐH hiện đại là một hệ thống tích hợp. Nó là việc tiến hóa từ HĐH tập trung tới HĐH mạng, HĐH phân tán và sau đó là hệ tự trị cộng tác, trong đó người dùng tiếp xúc với việc xây dựng các ứng dụng cộng tác lớn dựa trên các khối đã được cấu trúc hoàn hảo.

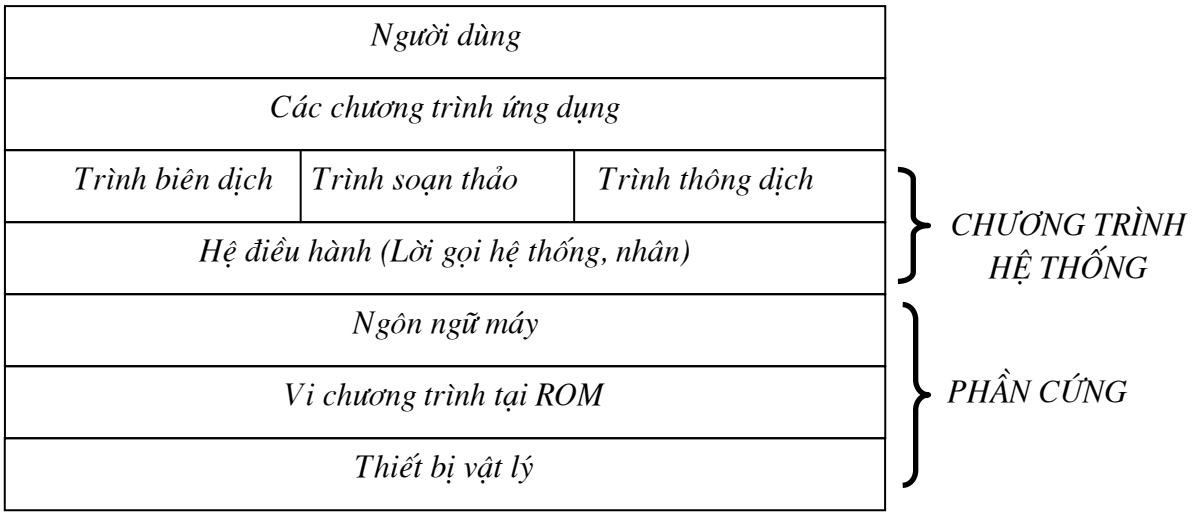
1.2. Tổng quan về hệ điều hành truyền thống

Như đã biết, HĐH truyền thống (còn được gọi là HĐH tập trung với đơn/đa bộ xử lý) chạy trên một máy tính là thế hệ HĐH đầu tiên, với độ kết dính chặt chẽ phần mềm - phần cứng trong đó mọi tài nguyên được chia xẻ một cách nội tại và truyền thông liên xử lý/liên QT được thực hiện qua hoặc chia xẻ bộ nhớ hoặc ngắt QT trực tiếp. Trong HĐH tập trung, hệ thống máy tính là tập trung: CPU (một hoặc nhiều) và bộ nhớ trong thỏa mãn một số tính chất nguyên thủy của chúng (ví dụ, tốc độ truy nhập của một CPU bất kỳ tới một địa chỉ bộ nhớ trong bất kỳ là đồng nhất ...). Coi rằng chỉ có duy nhất "một bộ CPU" cùng duy nhất "một bộ nhớ trong" và không hề quan tâm đến sự khác biệt thời gian truyền thông giữa các CPU hay giữa các bộ phận của bộ nhớ trong. Trong các HĐH truyền thống, chức năng hệ quản trị tài nguyên được nhấn mạnh hơn cho nên việc thiết kế chúng định hướng vào khai thác hiệu quả các tài nguyên phần cứng của hệ thống. Các bài toán điều khiển CPU (lập lịch), điều khiển bộ nhớ trong, điều khiển dữ liệu được đặc biệt chú ý. HĐH truyền thống được tiến hóa từ một chương trình đơn giản (cung cấp một giao diện người dùng và điều khiển vào - ra) tới một hệ đa người dùng/đa bài toán hoàn hảo với các yêu cầu về quản trị rất phức tạp đối với QT, bộ nhớ, file và thiết bị. Sự tiến hóa này được thể hiện trong bảng 1.2 mà các chức năng quản lý được đặt ra nhằm đáp ứng mỗi yêu cầu bổ sung.

Bảng 1.2. Chức năng chính của hệ điều hành tập trung

Yêu cầu hệ thống	*Chức năng quản lý*
Người dùng cá nhân	Giao diện người dùng, Điều khiển vào - ra, ngắt, điều khiển thiết bị
Vào - ra hiệu quả	Thiết bị vào - ra ảo, spooling
Chương trình lớn	Bộ nhớ ảo, phân trang hay phân segment
Đa người dùng	Đa chương trình và phân chia thời gian Lập lịch quá trình Điều khiển truy nhập và bảo vệ Chia xẻ file và điều khiển đồng thời
Đa bài toán (đa nhiệm)	Xử lý đồng thời Đồng bộ hóa quá trình, bế tắc Truyền thông liên quá trình

Có một lưu ý nhỏ về chính khái niệm *hệ điều hành* trong thời kỳ đánh dấu sự phát triển mạnh của HĐH truyền thống (vào khoảng những năm 1980). Trong nhiều tài liệu, đặc biệt là tài liệu về thiết kế HĐH, "hệ điều hành" được hiểu theo những nội dung đã được trình bày trên đây. Nhưng trong không ít các tài liệu khác, "hệ điều hành" được hiểu như bộ các chương trình hệ thống (xem hình 1.3) được cung cấp cho người sử dụng và ngoài những thành tố đã nói - tương ứng với *thành phần điều kiển*, HĐH còn có *thành phần ứng dụng* và *thành phần tiện ích*. Lý do chính của việc mở rộng nội dung khái niệm về HĐH như vậy liên quan đến sản phẩm kết quả cung cấp cho người sử dụng là một "bộ phần mềm hệ điều hành". Tuy nhiên, khi trình bày bản chất của HĐH, cách quan niệm này cũng nhất quán với cách quan niệm đã nói và nội dung trong giáo trình này nhất quán theo cách quan niệm như vậy.



Hình 1.3. Một cách nhìn khác về kiến trúc mức hệ thống máy tính

Tiếp theo trong mục dưới đây, chúng ta mô tả sơ lược quá trình tiến hóa của HĐH truyền thống.

1.2.0. Tiến hóa hệ điều hành truyền thống

a. Hệ điều hành đơn chương trình

HĐH đơn chương trình (HĐH dãy: serial OS) xuất hiện đầu tiên: chương trình của người dùng được xếp hàng để lần lượt được đưa vào bộ nhớ trong và chạy (thực hiện). Một chương trình sau khi được nạp từ dòng xếp hàng vào bộ nhớ trong được hệ thống (cùng toàn bộ tài nguyên) phục vụ từ khi chương trình bắt đầu chạy cho đến lúc chương trình kết thúc. Một chương trình được nạp vào bộ nhớ như vậy có thể được thực hiện với nhiều bộ dữ liệu. Chỉ khi chương trình này kết thúc thì mới nạp tiếp chương trình khác trong dòng đợi vào bộ nhớ trong. Trong hệ thống đơn chương trình thực chất không cần giải quyết bài toán điều khiển CPU (lập lich) vì CPU đã được dành riêng cho chương trình hiện tại.

Tuy nhiên, việc nạp chương trình và dữ liệu vào bộ nhớ trong làm việc lại liên quan đến thiết bị vào-ra đa dạng mà trong giai đoạn ban đầu phổ biến là vào bìa đục lỗ (thiết bị vào chuẩn) và ra máy in (thiết bị ra chuẩn). Và tới một thời điểm, ra đời CPU tốc độ cao, tốc độ nạp bìa cũng như tốc độ in ra không theo kịp với tốc độ của CPU, vì thế làm tăng thời gian nghỉ vô ích của CPU mà gây ra lãng phí. Đòi hỏi cần cải tiến nhằm tăng hiệu quả hoạt động. Một trong những cải tiến đối với HĐH đơn chương trình là hoạt động theo chế độ SPOOLING (Simultaneous Peripheral Operation OnLine), mà theo đó tất cả việc vào - ra đối với HĐH là làm việc với đĩa cứng còn vào - ra từ đĩa

cứng với các vật mang tin khác được đảm bảo bằng những ***cơ chế riêng***. Tốc độ của toàn bộ hệ thống được tăng lên đáng kể. Chế độ SPOOLING còn được sử dụng trong những HĐH đa chương trình xuất hiện sau này.

b. Hệ điều hành đa chương trình

Sự tiến bộ nhanh chóng của công nghệ máy tính dẫn tới dung lượng bộ nhớ tăng lên đáng kể (vượt xa dung lượng trung bình của các chương trình người dùng) và tốc độ CPU cũng tăng nhanh, chế độ hoạt động đa chương trình xuất hiện. Chế độ đa chương trình (multiprogramming) được phân loại theo hướng độc lập người dùng (chế độ mẻ) và hướng thân thiện người dùng (chế độ đa người dùng).

Đối với HĐH đa chương trình, tại mỗi thời điểm có thể có nhiều chương trình đồng thời có mặt ở bộ nhớ trong. Các chương trình này đều có nhu cầu được phân phối bộ nhớ và CPU để thực hiện. Như vậy, bộ nhớ, CPU, các thiết bị ngoại vi v.v. là các tài nguyên của hệ thống được chia xẻ cho các chương trình. Đặc điểm quan trọng cần lưu ý là các chương trình này phải được "bình đẳng" khi giải quyết các yêu cầu tài nguyên. Khái niệm *chương trình* nói trong chế độ đa chương trình được dùng để chỉ cả chương trình người dùng lẫn chương trình HĐH.

Khi so sánh với HĐH đơn chương trình, có thể nhận thấy ngay một điều là đối với một chương trình cụ thể thì trong chế độ đơn chương trình, chương trình đó sẽ kết thúc nhanh hơn (thời gian chạy ngắn hơn) so với khi nó chạy trong chế độ đa chương trình; nhưng bù lại, trong một khoảng thời gian xác định thì chế độ đa chương trình sẽ hoàn thiện được nhiều chương trình (giải được nhiều bài toán) hơn, do đó hiệu quả sử dụng máy tính cao hơn.

Một trong những tài nguyên quan trọng nhất của hệ thống máy tính là CPU. Việc chia xẻ CPU là một trong những dạng điển hình của việc chia xẻ tài nguyên. Tính chất chia xẻ CPU lại phân lớp các HĐH đa chương trình thành các lớp con: HĐH hoạt động theo *chế độ mẻ (batch)* và HĐH hoạt động theo *chế độ phân chia thời gian (time shared)*.

• *Hệ điều hành hoạt động theo chế độ mẻ*

Đây là loại HĐH định hướng tới mục tiêu làm *cực đại số lượng các bài toán được giải quyết* trong một khoảng đơn vị thời gian (có nghĩa là trong một khoảng đơn vị thời gian thì hướng mục tiêu vào việc hoàn thiện được càng nhiều chương trình càng tốt). Ở nước ta những năm trước đây, các máy tính EC-1022, EC-1035 (HĐH OS), IBM 360/40-50 (HĐH DOS) phổ biến hoạt động theo chế độ mẻ. Trong HĐH chế độ mẻ, cách thức điều khiển CPU điển hình là một chương trình ở trạng thái sẵn sàng sẽ được chọn thực hiện (được phân phối CPU) khi chương trình đang chạy phải ngừng vì nó cần đến một tài nguyên khác CPU.

Các HĐH theo chế độ mẻ lại có thể phân biệt thành hai loại điển hình là MFT và MVT: sự phân biệt chúng theo cách điều khiển bộ nhớ trong.

MFT: Multiprogramming with Fixed number of Tasks

Khi hệ thống làm việc, đã quy định sẵn một số lượng cố định các bài toán đồng thời ở bộ nhớ trong: Bộ nhớ trong được chia thành một số vùng nhớ cố định, các vùng này có biên cố định mà mỗi vùng được dùng để chứa một chương trình tại một thời điểm. Mỗi chương trình người dùng chỉ được đưa vào một vùng nhớ xác định tương ứng với chương trình đó. Một chương trình chỉ có thể làm việc trong giới hạn của vùng bộ nhớ trong đang chứa nó: chương trình đó tồn tại trong vùng bộ nhớ tương ứng trong suốt thời gian nó được thực hiện trong máy tính, kể từ lúc bắt đầu cho tới lúc kết thúc.

MVT: Multiprogramming with Variable number of Tasks

Khác với chế độ MFT, trong chế độ MVT, bộ nhớ trong không bị chia sẵn thành các vùng, việc nạp chương trình mới vào bộ nhớ trong còn được tiếp diễn khi mà bộ nhớ trong còn đủ để chứa thêm chương trình.

Có thể quan niệm rằng trong chế độ MFT bộ nhớ trong được phân thành các vùng có vách ngăn cố định, còn trong chế độ MVT, không có vách ngăn sẵn, mỗi khi chương trình được nạp vào mới hình thành một vách ngăn tạm thời. Nếu chỉ gặp các chương trình đòi hỏi ít bộ nhớ thì theo chế độ MVT, số lượng chương trình đồng thời có mặt trong bộ nhớ nhiều lên.

• *Chế độ phân chia thời gian (Time Shared System: TSS)*

Chế độ phân chia thời gian là chế độ hoạt động điển hình của các HĐH đa người dùng (*multi-users*). HĐH hoạt động theo chế độ này định hướng phục vụ trực tiếp người dùng khi chương trình của người dùng đó đang thực hiện, làm cho giao tiếp của người dùng với máy tính là hết sức thân thiện. Liên quan đến HĐH hoạt động theo chế độ này là các khái niệm lượng tử thời gian, bộ nhớ ảo v.v.

Trong hệ TSS, tại cùng thời điểm có nhiều người dùng đồng thời làm việc với máy tính: Mỗi người làm việc với máy tính thông qua một trạm cuối (terminal) và vì vậy, hệ thống đã cho phép máy tính thân thiện với người dùng.

Khác với cách thức điều khiển CPU trong chế độ mẻ, HĐH phân phối CPU lần lượt cho từng chương trình người dùng, mỗi chương trình được chiếm giữ CPU trong một khoảng thời gian như nhau (khoảng thời gian đó được gọi là lượng tử thời gian: time quantum): có thể thấy phổ biến về lượng tử thời gian điển hình là khoảng 0,05s. Máy tính làm việc với tốc độ cao, chu kỳ quay lại phục vụ cho từng chương trình người dùng là rất nhanh so với giác quan của người dùng, và vì vậy, mỗi người dùng đều có cảm giác rằng mình đang chiếm hữu toàn bộ tài nguyên hệ thống.

Điều khiển bộ nhớ trong của chế độ đa người dùng có nhiều khác biệt bản chất so với chế độ mẻ. Bộ nhớ trong luôn chứa chương trình của mọi người dùng, vì vậy xảy ra tình huống toàn bộ bộ nhớ trong không đủ để chứa tất cả chương trình người dùng hiện đang thực hiện; vì vậy, đối với HĐH TSS nảy sinh giải pháp sử dụng bộ nhớ ảo: sử dụng đĩa từ như vùng mở rộng không gian nhớ của bộ nhớ trong.

HĐH UNIX (và Linux) là HĐH đa người dùng điển hình.

Có thể nhận xét rằng, *tính quản trị tài nguyên* được nhấn mạnh trong HĐH mẻ và *tính chất máy tính ảo* đã được quan tâm hơn trong HĐH đa người dùng.

c. Hệ điều hành thời gian thực

Nhiều bài toán trong lĩnh vực điều khiển cần được giải quyết không muộn hơn một thời điểm nhất định, và vì vậy, đối với các máy tính trong lĩnh vực đó cần HĐH thời gian thực (RT: Real Time). Trong hệ thời gian thực, mỗi bài toán được gắn với một thời điểm tới hạn (tiếng Anh là *deadtime*) và bài toán phải được giải quyết không muộn hơn thời điểm đã cho đó: Nếu bài toán hoàn thiện muộn hơn thời điểm đó thì việc giải quyết nó trở nên không còn ý nghĩa nữa. Hệ thời gian thực có thể được coi như một trường hợp của hệ đa chương trình hoạt động theo chế độ mẻ có gắn thêm thời điểm kết thúc cho mỗi bài toán.

d. Hệ điều hành kết hợp

Các nhà thiết kế HĐH hiện đại cũng chọn lựa việc thiết kế HĐH có khả năng khởi tạo hoạt động được theo một trong một số chế độ hoạt động của HĐH đã nói trên đây. Chẳng hạn, HĐH OS cho hệ thống máy EC hoặc IBM có thể hoạt động hoặc theo chế

độ mẻ (MFT, MVT) hoặc theo chế độ phân chia thời gian (SYS); hoặc HĐH LINUX hoạt động theo chế độ đơn người dùng (với superuser) hoặc chế độ đa người dùng (với các người dùng khác). Kiểu hệ điều hành như vậy được quan niệm là kết hợp nội dung của nhiều loại hệ điều hành (Combination Operating System).

e. Hệ thống đa xử lý

Hệ thống nhiều CPU

Hiện nay, từ tốc độ phát triển nhanh của công nghệ, máy tính ngày càng được phổ dụng trong xã hội. Mức độ thâm nhập của máy tính vào cuộc sống càng cao thì yêu cầu nâng cao năng lực của máy tính lại ngày càng trở nên cấp thiết. Bộ nhớ chính ngày càng rộng lớn; đĩa từ có dung lượng càng rộng, tốc độ truy nhập ngày càng cao; hệ thống thiết bị ngoại vi càng phong phú, hình thức giao tiếp người-máy ngày càng đa dạng. Như đã nói, CPU là một tài nguyên thể hiện chủ yếu nhất năng lực của hệ thống máy tính, vì vậy một trong những vấn đề trọng tâm nhất để tăng cường năng lực của hệ thống là tăng cường năng lực của CPU. Đối với vấn đề này, nảy sinh các giải pháp theo hai hướng:

Giải pháp *tăng cường năng lực của một CPU riêng* cho từng máy tính: công nghệ vi mạch ngày càng phát triển vì vậy năng lực của từng CPU cũng ngày nâng cao, các dự án vi mạch VLSI với hàng triệu, hàng chục tiệu transitor được triển khai. Tuy nhiên giải pháp này cũng nảy sinh những hạn chế về kỹ thuật: tốc độ truyền thông tin không vượt qua tốc độ ánh sáng; khoảng cách gần nhất giữa hai thành phần không thể giảm thiểu quá nhỏ v.v.

Song song với giải pháp tăng cường năng lực từng CPU là giải pháp *liên kết nhiều CPU* để tạo ra một hệ thống chung có năng lực đáng kể: việc xử lý song song tạo ra nhiều lợi điểm. Thứ nhất, chia các phần nhỏ công việc cho mỗi CPU đảm nhận, năng suất tăng không chỉ theo tỷ lệ thuận với một hệ số nhân mà còn cao hơn do không mất thời gian phải thực hiện những công việc trung gian.

Thứ hai, giải pháp này còn có lợi điểm tích hợp các hệ thống máy đã có để tạo ra một hệ thống mới với sức mạnh tăng gấp bội.

Chúng ta khảo sát một số nội dung chọn giải pháp đa xử lý theo nghĩa một hệ thống tính toán được tổ hợp không chỉ một CPU mà nhiều CPU trong một máy tính (*hệ đa xử lý tập trung*) hoặc nhiều máy tính trong một hệ thống thống nhất. Gọi chung các hệ có nhiều CPU như vậy là *hệ đa xử lý*.

Phân loại các hệ đa xử lý

Có một số cách phân loại các hệ đa xử lý:

• Phân loại theo *vị trí đặt các CPU*: tập trung hoặc phân tán.

Các siêu máy tính (supercomputer) là các ví dụ về hệ đa xử lý tập trung. Đặc trưng của hệ thống này là các CPU được liên kết với nhau trong một máy tính duy nhất đảm bảo độ kết dính phần cứng chặt. Ví dụ về hệ đa xử lý phân tán là các hệ thống tính toán phân tán dựa trên mạng máy tính với độ kết dính phần cứng lỏng.

• Phân loại theo *đặc tính của các CPU* thành phần: hệ đa xử lý thuần nhất hoặc hệ đa xử lý không thuần nhất v.v. Một ví dụ quen thuộc về hệ không thuần nhất là thiết bị xử lý trong máy vi tính gồm CPU xử lý chung và CPU xử lý dấu phảy động. Siêu máy tính ILLIAC-IV gồm nhiều CPU có đặc trưng giống nhau là một ví dụ về hệ thuần nhất.

• Cách phân loại điển hình là dựa theo kiểu các CPU thành phần *tiếp nhận và xử lý dữ liệu* trong một nhịp làm việc. Cách phân loại này bao gồm cả máy tính đơn xử lý thông thường:

- Đơn chỉ thị, đơn dữ liệu (SISD: Single Data Single Instruction) được thể hiện trong máy tính thông thường; Mỗi lần làm việc, CPU chỉ xử lý "một dữ liệu" và chỉ có một chỉ thị (instruction, câu lệnh) được thực hiện. Đây là máy tính đơn xử lý.

- Đơn chỉ thị, đa dữ liệu (SIMD: Single Instruction Multiple Data):

Các bộ xử lý trong cùng một nhịp làm việc thực hiện chỉ cùng một chỉ thị. Ví dụ như phép cộng hai vector cho trước: Các CPU thành phần đều thực hiện các phép cộng theo đối số tương ứng tại mỗi CPU; sau đó, chọn tiếp chỉ thị mới để tiếp tục công việc. Thông thường, hệ thống có bộ phận điều khiển riêng cho việc chọn chỉ thị và mọi CPU thành phần cùng thực hiện chỉ thị đó (bộ xử lý ma trận).

- Đa chỉ thị, đơn dữ liệu (MISD: Multiple Instruction Single Data):

Trong các máy tính thuộc loại này, hệ thống gồm nhiều CPU, các CPU liên kết nhau tuần tự: output của CPU này là input của CPU tiếp theo (Bộ xử lý vector). Các CPU kết nối theo kiểu này được gọi là kết nối "dây chuyền".

- Đa chỉ thị, đa câu lệnh (MIMD):

Mỗi CPU có bộ phân tích chương trình riêng; chỉ thị và dữ liệu gắn với mỗi CPU: nhịp hoạt động của các CPU này hoàn toàn "độc lập nhau".

1.2.1. Cấu trúc hệ điều hành truyền thống

<table>
<tr><td>Các ứng dụng</td><td>Kế toán ...</td><td colspan="2">Văn phòng</td><td>Sản xuất</td></tr>
<tr><td>Các hệ thống con</td><td colspan="2">Môi trường lập trình</td><td colspan="2">Hệ thống cơ sở dữ liệu</td></tr>
<tr><td>Các tiện ích</td><td>Bộ biên dịch</td><td colspan="2">Thông dịch lệnh</td><td>Thư viện</td></tr>
<tr><td>Các dịch vụ hệ thống</td><td>Hệ thống File</td><td>Quản lý bộ nhớ</td><td colspan="2">Bộ lập lịch</td></tr>
<tr><td>Nhân</td><td colspan="4">CPU đa thành phần, kiểm soát ngắt, điều khiển thiết bị, đồng bộ nguyên thủy, truyền thông liên quá trình</td></tr>
</table>

Hình 1.4 Phân mức các thành phần hệ điều hành theo chiều dọc và chiều ngang

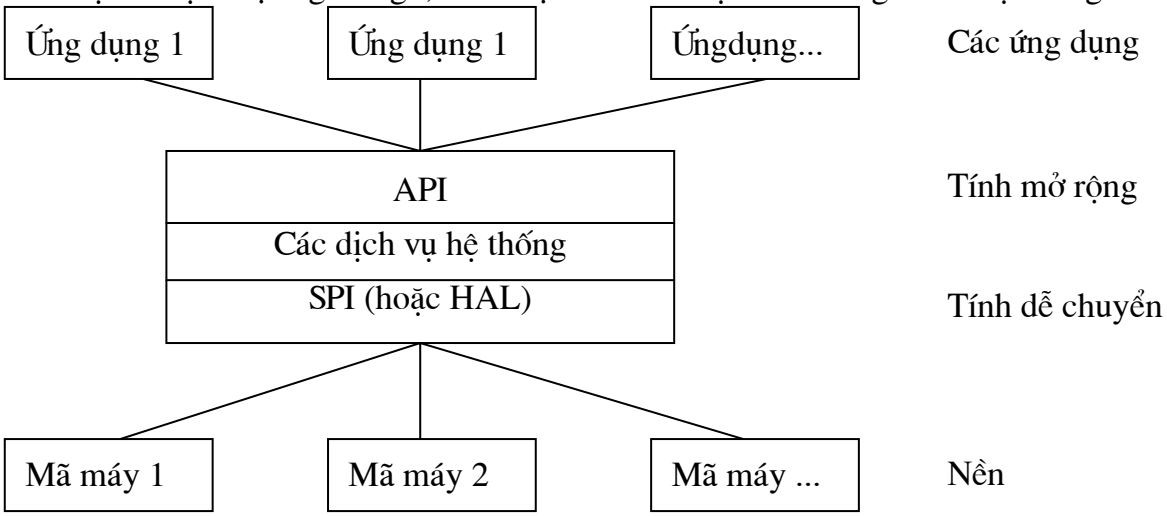
HĐH là bộ phần mềm lớn có kích thước từ hàng nghìn tới hàng triệu dòng mã lệnh, cho nên khi thi hành cần thiết phải kiến trúc phần mềm HĐH từ các môđun dễ dàng quản lý (phù hợp với phương pháp chung phát triển phần mềm). Kỳ vọng một kết quả thiết kế là cung cấp các giao diện được xác định tường minh giữa các môđun và cách đặt các ràng buộc cho tương tác môđun. Hai cách tiếp cận truyền thống thông dụng phân hoạch các môđun là phân hoạch ngang và phân hoạch dọc. Phân hoạch dọc dựa trên khái niệm mức, theo đó phân chia các môđun thành nhóm theo các mức khác nhau với ràng buộc là chỉ cho phép tương tác giữa các môđun thuộc hai mức liền kề. Đây là giao diện *một vào - một ra* giữa các mức. Các môđun trong từng mức (phân hoạch ngang) lại được tổ chức thành các thành phần lớn rời rạc nhau, mỗi thành phần như thế lại có thể được tinh chế tiếp theo tổ hợp phân hoạch ngang hoặc dọc (hình 1.4). Kiểu điển hình của môđun là tập dòng lệnh thi hành một dịch vụ hệ thống riêng biệt. Trong một hệ thống hướng đối tượng, các môđun cần được thi hành như một đối tượng với các thao tác (hoàn toàn xác định) trên mỗi đối tượng dịch vụ thành phần. Môđun hóa dựa trên đối tượng là tốt hơn so với mã hóa. Mọi tài nguyên, bao gồm cả file và QT, cần được tiếp cận như đối tượng dữ liệu mà thể hiện vật lý của chúng phải được che giấu bằng các cấu trúc dữ liệu trừu tượng. Hoạt động (thực hiện) của môđun được giới

hạn bằng một tập các thao tác và luật hình thức gán cho đối tượng. Môđun hướng đối tượng cung cấp hàng loạt lợi thế, trong đó có tính đồng nhất truy nhập và bảo vệ. Vì đồng nhất, chúng dễ dàng biến đổi và do đó làm tăng tính khả chuyển của hệ thống.

Tính khả chuyển của HĐH còn tăng lên khi tách các mã phụ thuộc-máy từ hệ thống. Đa phần các phần mềm HĐH (các dịch vụ hệ thống) là độc lập phần cứng. Từ đó, hệ thống cần được cấu trúc theo cách mà phần phụ thuộc-máy được giữ ở *mức tối thiểu nhất* và *tách rời khỏi các dịch vụ hệ thống*. Cách tiếp cận nhân tối thiểu này làm giảm bớt độ phức tạp về tính khả chuyển hệ thống từ kiểu kiến trúc máy tính này sang kiểu kiến trúc máy tính khác vì chỉ có nhân mới phải viết lại. Các chức năng điển hình được thi hành trong nhân tối thiểu bao gồm: tính đa thành phần của các bộ xử lý với hỗ trợ đa chương trình, kiểm soát ngắt, điều khiển thiết bị, (dịch vụ) nguyên thuỷ đồng bộ QT (ĐBQT), các phương tiện truyền thông liên QT (TTLQT, tiếng Anh Interprocess Communication, IPC). Cấu trúc nhân thường *nguyên khối* theo nghĩa không còn phân hoạch ngang hoặc dọc được nữa mà chỉ là môđun hóa theo mã. Cấu trúc này đạt được do nhân đã được tối thiểu nhất. Triết lý thiết kế này là hiệu quả cho phép chú ý tới cách liên kết nhân hơn là cấu trúc nhân. Hình 1.4 thể hiện các khái niệm môđun hóa và cấu trúc hóa với một số thành phần trong mỗi mức của phần mềm hệ thống.

Cấu trúc HĐH được nâng cao theo mô hình Client/Server. Mô hình Client/Server là một mô hình lập trình khuôn mẫu. Theo mô hình này, *lời gọi hệ thống* từ các chương trình ứng dụng yêu cầu các *dịch vụ HĐH* giống như yêu cầu QT khách trực tiếp tới QT phục vụ. Chúng được thi hành gián tiếp thông qua nhân HĐH. Lời gọi hệ thống chia xẻ một lối vào nhất quán tới hệ thống. Cơ chế này làm đơn giản hóa tương tác tới HĐH và cho phép người thiết kế hệ thống chuyển thêm nhiều dịch vụ hệ thống tới mức cao hơn (trong nhiều trường hợp tới không gian QT người dùng) và kết quả được nhân nhỏ hơn và dễ quản lý hơn. Mô hình Client/Server là cách tự nhiên mô tả các tương tác giữa các QT trong hệ phân tán trong khi chuyển thông điệp (một khái niệm quan trọng) chỉ có nghĩa chuyển vận dữ liệu trong các thực thể truyền thông.

Sự phân biệt giữa chương trình ứng dụng với chương trình hệ thống thường mơ hồ. Chương trình trong nhân và dịch vụ hệ thống là phần mềm hệ thống (xem hình 1.4). Tuy nhiên, theo một quan niệm khác (như đã được trình bày trong hình 1.3), chương trình hệ thống còn bao gồm cả trình biên dịch, trình soạn thảo hệ thống, trình thông dịch [3]. Chính bởi lý do đó mà người viết trình biên dịch cũng tự coi họ là những người lập trình hệ thống. Tuy nhiên, trình biên dịch theo quan điểm của HĐH được xem là một ứng dụng. Mặt khác, phần mềm cơ sở dữ liệu là một ứng dụng đối với trình biên dịch hoặc một ngôn ngữ, đến lượt mình nó lại là chương trình hệ thống đối với

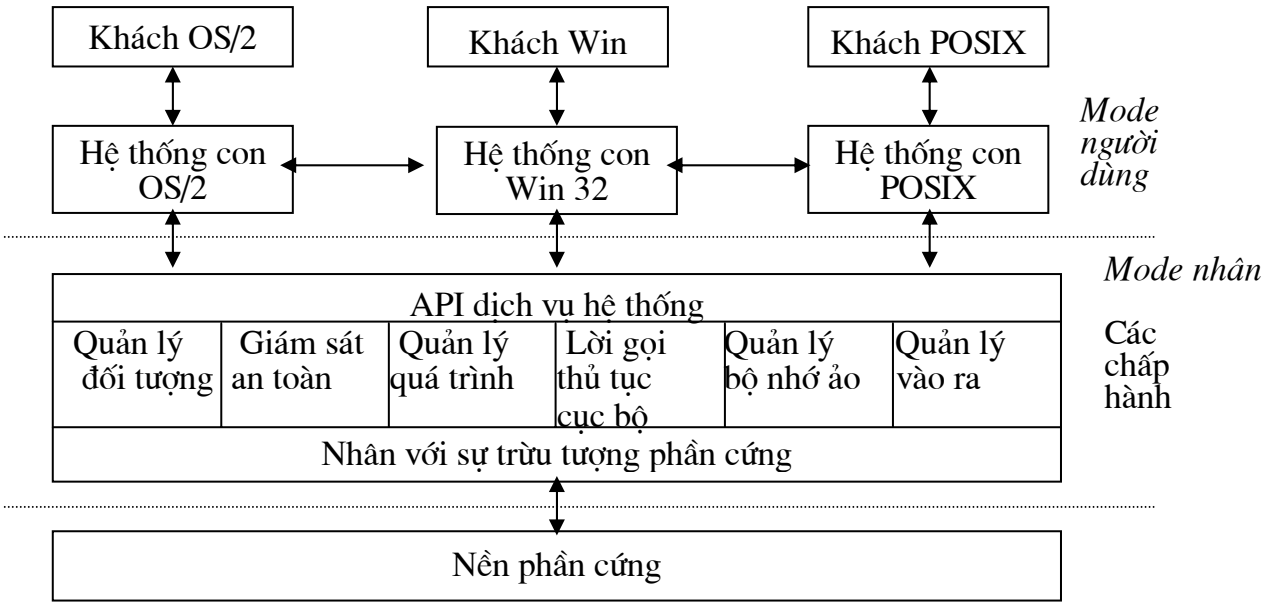


Hình 1.5. Các mức API và SPI

người dùng. Kiến trúc này có thể phát triển thêm một vài mức. Người dùng nhìn hệ thống qua một hệ thống con được đặc trưng bằng các dịch vụ cung cấp cho họ. Mối quan hệ giữa chương trình hệ thống và chương trình ứng dụng là gần gũi.

1.2.2. Hệ thống con và vi nhân

Nhân tối thiểu vạn năng mà dựa trên nó, các dịch vụ HĐH chuẩn được thi hành nhằm hỗ trợ các hệ thống con hướng ứng dụng được gọi là vi nhân. Kiến trúc vi nhân bao gồm một nhân tối thiểu phụ thuộc nền phần cứng và một tập các thi hành độc lập phần cứng (dịch vụ hệ thống) bằng Bộ giao diện trình ứng dụng (API: Application Program Interface) hoàn toàn xác định. Điều ý nghĩa của khái niệm vi nhân ở chỗ nó cung cấp một môi trường chứa các điều kiện cần và đủ để cấu trúc HĐH hoặc hệ thống con đáp ứng nhu cầu bất kỳ với ít công sức nhất. Nhân, mặc dù phụ thuộc phần cứng, nhưng được cấu trúc với độ trừu tượng phần cứng để dễ dàng được ghi lại mã máy khi được mang chuyển tới một nền khác. Cấu trúc bổ sung này được gọi là Mức trừu tượng phần cứng (HAL: Hardware Abstraction Layer) hoặc Giao diện cung cấp dịch vụ (SPI: Service Provider Interface) khi được sử dụng trong mô đun phần mềm mức trên. Như trình bày tại hình 1.5 thì API cung cấp tính mở rộng cho các ứng dụng mức cao còn SPI (hoặc HAL) đề cao tính khả chuyển cho nền tảng mức thấp.



Hình 1.6. Kiến trúc hệ thống Windows NT

Các dịch vụ hệ thống là đồng hạng môđun và đầy đủ vì vậy chúng được dùng như một cơ chế để hỗ trợ lớp rộng lớn các ứng dụng. Người thiết kế phần mềm chỉ cần biết các giao diện chuẩn tới các môđun dịch vụ thi hành và có thể chọn một tập con của chúng theo đòi hỏi. Kiến trúc như vậy rất thuyết phục vì tính môđun hóa, dễ mang chuyển và khả năng tiếp thị (môđun thi hành có thể thuộc bản quyền của nhà cung cấp hệ thống khác). Ngành công nghiệp đã có một vài cố gắng hướng tới một kiến trúc vi nhân chung, đáng kể nhất là Microkernel của IBM và Windows NT của Microsoft.

Windows NT được trình bày trong hình 1.6 như một ví dụ của kiến trúc vi nhân, trong kiến trúc này mỗi khách thấy máy tính theo một môi trường tính toán khác nhau (OS/2, Win32, POSIX) được biểu diễn bởi API trong hệ thống con. Mỗi hệ thống con có không gian địa chỉ lôgic riêng của mình, có thể được cô lập và bảo vệ đối với các hệ thống con khác. Lời gọi hệ thống con dựa trên API dịch vụ hệ thống và như vậy được thi hành một cách độc lập với việc thực hiện tại mức nhân và mức phần cứng. Như một

chọn lựa, hệ thống con có thể tương tác gián tiếp với dịch vụ hệ thống qua hệ thống con Win32, hệ này hỗ trợ phương tiện lập trình window. Kiến trúc tại hình vẽ đạt được mọi khái niệm kiến trúc hệ thống như đã thảo luận: môđun hóa, phân mức, mô hình Client/Server, mô hình đối tượng, và nhân tối thiểu. Hầu hết các HĐH hện đại theo đuổi triết lý thiết kế như vậy với một vài khác biệt nhỏ khi thi hành.

1.2.3. Các chức năng quản tri

HĐH có chức năng quản trị tài nguyên. Mỗi tài nguyên trong hệ thống máy tính nói chung thuộc vào một trong bốn lớp: Bộ xử lý/quá trình, bộ nhớ, I/O và dữ liệu/file. Mô tả một cách tóm tắt các vấn đề cơ sở của HĐH truyền thống. Tóm tắt này như là những thông tin nền tảng cho thảo luận HĐH mạng, HĐH phân tán và hệ tự trị cộng tác.

• *Quản trị Bộ xử lý/Quá trình* ở mức hệ thống thấp nhất là cung cấp ánh xạ (lập lịch: scheduling) các bộ xử lý tới các QT, hoặc ngược lại. Để thuận tiện cho việc mở rộng đa người dùng và đa bài toán (đa nhiệm), cần tới tính đa thành phần-không gian bộ nhớ (nơi đặt QT) và tính đa thành phần-thời gian các bộ xử lý (nơi QT thực hiện). Thi hành tính đa thành phần như vậy thông qua tính đa chương trình và phân chia thời gian được cơ chế kiểm soát ngắt đầy đủ hỗ trợ. Tại mức cao, việc thi hành là trong suốt tới các QT thực hiện đồng thời. Người dùng chỉ quan tâm tới việc phối hợp tương tác các QT đồng thời. Các tương tác đòi hỏi ĐBQT và TTLQT. Trong hơn hai chục năm trở lại đây, hàng loạt phương pháp ĐBQT được đề xuất nhằm giải quyết bài toán đồng bộ nhờ loại trừ ràng buộc và kết hợp trạng thái.

Hầu hết các tiếp cận cơ sở dùng lời gọi hệ thống đặc biệt thao tác trên các biến kiểu dữ liệu *semaphore*. Do được hệ thống hỗ trợ khả năng kết khối QT, các thao tác nguyên tử trên semaphore (yêu cầu tài nguyên P(s) / giải phóng tài nguyên V(s)) cho phép phối hợp các QT tương tác. Các tiếp cận khác (không dùng semaphore) gắn năng lực đồng bộ vào ngôn ngữ lập trình nhờ hoặc là biến dạng cấu trúc điều khiển (chẳng hạn, *khoảng tới hạn có điều kiện* - Condition Critical Region, ghi tắt CCR. Khái niệm cơ sở là *khoảng tới hạn* - Critical Region, ghi tắt là CR) hoặc là bổ sung kiểu dữ liệu trừu tượng mới (chẳng hạn, *bộ giám sát* - monitor). Ngoài ra, một cách thức ĐBQT khác dựa theo cách lệnh vào-ra, chẳng hạn *Bộ các quá trình tuần tự truyền thông* (Communicating Sequential Processes: CSP). Đây là cách tiếp cận tổng quát hơn theo kiểu lời gọi thủ tục và dẫn dắt *điểm hẹn* (rendezvous) trong ngôn ngữ lập trình Ada. Tiến thêm một bước mới, cho phép đặc tả dãy các điều khiển thực hiện đồng thời trong một chương trình mà không cần dùng các nguyên thủy đồng bộ một cách tường minh, như được thi hành trong *biểu thức đường đi* (Path Expression: PE). Người ta chỉ ra rằng, mọi tiếp cận được đề xuất cho bài toán đồng bộ có thể được thi hành với sự thỏa hiệp giữa hiệu quả và năng lực diễn tả lời giải bài toán. Các phương pháp đồng bộ truyền thống sẽ được mô tả sơ lược tại chương 3).

Các phương tiện TTLQT được phát triển song hành với ĐBQT. Trong HĐH tập trung, TTLQT xuyên qua *chia xẻ bộ nhớ* dường như là giải pháp dễ dàng. Tuy nhiên, chia xẻ bộ nhớ vi phạm giả thiết cơ bản là các QT không đồng bộ và nhìn chung là không chia xẻ không gian địa chỉ chung. Lựa chọn còn lại là *truyền thông* thông qua chuyển thông điệp (ghi tắt CTĐ, *message passing*). Ưu điểm của CTĐ do nó là một phần bản chất của hệ phân tán và như vậy có thể đưa việc phát triển HĐH tập trung thích hợp với việc phát triển hệ phân tán.

Điều rất có giá trị chính là mối quan hệ thân thiết giữa ĐBQT và truyền thông QT (TTQT). TTQT đòi hỏi một số giả thiết nền tảng từ ĐBQT, chẳng hạn như đồng bộ gửi và nhận dữ liệu. Với các dịch vụ nguyên thủy của truyền thông QT, cấu trúc đồng bộ mức cao có thể được thi hành chỉ dựa trên các TTQT nguyên thủy. Khởi đầu từ nguyên

thủy đồng bộ và truyền thông cũng dẫn đến việc phát triển ngôn ngữ đồng thời: các ngôn ngữ cho phép đặc tả được QT đồng thời, đồng bộ và TTLQT. Ngôn ngữ đồng thời và đồng bộ/truyền thông trong hệ phân tán được bàn luận tương ứng trong chương 3 và chương 4.

Cùng với ĐBQT và TTQT, quản trị QT còn có chức năng lập lịch. Quá trình đang *sẵn sàng* (ready) hoặc ở *dòng xếp hàng* (waiting sequence) cần được lập lịch lại để thực hiện khi tài nguyên đã sẵn sàng hoặc điều kiện nào đó được thỏa mãn. Rất nhiều chiến lược được dùng nhằm đạt được hàm mục tiêu, chẳng hạn tối thiểu thời gian chuyển lịch hoặc tối đa thông lượng hệ thống (system throughput). Lập lịch bài toán (task, hoặc quá trình - process) cho máy đơn-đa xử lý là một vấn đề nghiên cứu thao tác cổ điển. Ứng dụng lập lịch bài toán vào hệ phân tán là phức tạp do tồn tại đa máy tính và tổng phí (overhead) truyền thông buộc phải tính đến trong lập lịch. Tồn tại hai kiểu lập lịch: *Lập lịch QT tĩnh* dựa trên mô hình quan hệ đi trước và *chia xẻ tải động quá trình* dựa trên mô hình quan hệ phụ thuộc quá trình. Quan hệ đi trước biểu diễn các QT buộc phải đồng bộ như thế nào, trong khi đó quan hệ phụ thuộc chỉ cho biết dấu hiệu tương tác giữa các QT. Hai kiểu lập lịch này biểu diễn độ hiểu biết khác nhau về mối tương tác giữa các QT trong đồng bộ và truyền thông. Lập lịch tĩnh, chia xẻ động và cân bằng tải được trình bày trong chương 5.

• *Quản trị thiết bị vào - ra* là trách nhiệm chặt chẽ của hệ thống các thiết bị gắn kết vật lý. Nhằm giảm bớt độ phức tạp khi thiết kế hệ thống theo tính phụ thuộc máy, kiến trúc hệ thống của bộ xử lý thường được tách hoàn toàn khỏi tính chi tiết thiết bị vào-ra. Bộ xử lý cung cấp một giao diện chung tới tất cả thiết bị và căn cứ theo giao diện chung đó, nhà chế tạo thiết bị vào-ra phát triển thiết bi điều khiển thiết bị vào-ra và trình điều khiển phần mềm để tích hợp vào hệ thống. Theo hướng trừu tượng, các thiết bị vào-ra chỉ là bộ ghi nhớ: Một số cho phép đọc và ghi (chẳng hạn, đĩa từ), một số khác chỉ cho phép đọc (chẳng hạn, bàn phím) và một số khác nữa chỉ cho phép ghi (chẳng hạn, máy in). Theo quan điểm của HĐH, thích hợp nhất coi tất cả thiết bị vào-ra là file lôgic. File lôgic biểu diễn thiết bị vật lý được gọi là thiết bị ảo. Các QT chỉ thao tác trên các file và hệ thống chịu trách nhiệm diễn giải file này tới thiết bị vật lý.

Người ta sử dụng một số kỹ thuật nhằm tăng tốc thao tác vào-ra, đáng kể nhất là hai khái niệm *spooling* và *buffering*. Spooling (như đã được giới thiệu tại trang 10) làm thuận tiện chia xẻ các thiết bị vào - ra, còn buffer (bộ đệm) căn bản được dùng để dàn xếp sự khác nhau về tốc độ làm việc giữa thiết bị vào-ra chậm và bộ xử lý nhanh. Buffer có thể được thi hành ở nhiều mức phần mềm khác nhau, chẳng hạn như hệ thống file, trình điều khiển thiết bị, và trong một số trường hợp ở ngay trong thiết bị điều khiển vào-ra. Hai thiết bị vào - ra quan trọng nhất là ổ đĩa và trạm cuối. Đĩa tốc độ cao và dung lượng rộng (vài trăm gigabytes) là rất thông dụng. Đĩa dung lượng cao đóng vai trò đáng kể trong việc thiết kế phần mềm lớn. Trạm cuối bản đồ - bộ nhớ tạo nên sự thi hành việc hỗ trợ các cửa sổ (windows) tại trạm cuối. Cửa sổ được khởi hành như một bàn giao tiếp ảo (virtual console) đơn giản. Với các chức năng bổ sung như một giao diện người dùng đồ họa và các cửa sổ đa tương tác, windows được tiến hóa thành giao diện đang phát triển một cách thăng hoa đối với hệ thống con và sẽ trở thành một máy tính ảo như trường hợp HĐH Windows 95.

Một vấn đề đáng quan tâm liên quan tới quản trị vào-ra là bế tắc (deadlock). Bế tắc nảy sinh trong hệ thống do định vị sai tài nguyên khi có một tập QT không ưu tiên (nonpreemptable) mà mỗi từ chúng giữ tài nguyên lại đòi hỏi tài nguyên từ QT trong tập đó, tạo ra một chu trình xâu QT không thể tháo rời. "Tài nguyên" có thể là thiết bị vật lý và (chung hơn) là các buffer và các điều kiện. Việc phòng ngừa, thoát ra, và phát

hiện bế tắc đã được nghiên cứu rộng rãi. Tuy nhiên, phát hiện và giải quyết bế tắc phân tán hiện vẫn đang là vấn đề mở.

• *Quản lý bộ nhớ* bao gồm việc *phân phối - phân phối lại* bộ nhớ và ánh xạ không gian chương trình lôgic vào bộ nhớ vật lý. Mục tiêu căn bản là bảo đảm tận dụng cao bộ nhớ và cung cấp bộ nhớ ảo hỗ trợ chương trình lớn, đặc biệt là các chương trình có kích thước vượt kích thước bộ nhớ vật lý. Hầu hết hệ thống máy tính hiện nay đều sử dụng các kỹ thuật điều khiển trang (paging)/ điều khiển segment (segmentation) khi thi hành bộ nhớ ảo. Thi hành bộ nhớ ảo đòi hỏi phần cứng bổ sung, thường được gọi là đơn vị quản lý bộ nhớ (memory managment unit). Cả trang và segment đều là các cơ chế phân phối bộ nhớ rời rạc. Sự khác nhau chính giữa hai cơ chế này phân chương trình theo trang vật lý và theo segment logic. HĐH hiện đại thi hành bộ nhớ ảo theo cơ chế tổ hợp hai cơ chế này. Do không phải tất cả các trang và segment đồng thời nằm trong bộ nhớ trong, nên cần điều tiết những chỉ dẫn tới dữ liệu và chỉ thị (lệnh) mới khi thực hiện một chương trình. Nhiều thuật toán thay trang được đề xuất nhằm rút gọn tần số lỗi trang. Hiệu suất của chiến lược thay trang phụ thuộc mạnh vào cách thực hiện chương trình tại khoảng thời gian đã cho bất kỳ. Định hướng không gian và thời gian được mô tả tổng quát trong chương trình có ảnh hưởng đáng kể khi chọn thuật toán thay trang.

Bộ nhớ ảo là giải pháp nhằm giải quyết sự khác nhau về kích thước và tốc độ giữa bộ nhớ đĩa chậm tương đối và bộ nhớ vật lý nhanh hơn. Tồn tại vấn đề tương tự khi bộ nhớ tốc độ cao (cache) được dùng như bộ đệm giữa bộ xử lý và bộ nhớ chính. Quá trình buffer này chỉ đòi hỏi phải ánh xạ địa chỉ vật lý (được gọi là caching) mà thông thường được quan tâm theo hướng kiến trúc hơn là vấn đề của HĐH. Chọn lựa thuật toán thay trang, ảnh hưởng của cỡ trang và segment, ảnh hưởng của phân phối bộ nhớ, caching và liên kết cache là một số vấn đề của quản trị bộ nhớ.

Trong HĐH tập trung, bộ nhớ chia xẻ cho giá trị là tính đơn giản đối với truyền thông và tương tác QT. Nhiều thuật toán được phát triển cho bộ nhớ chia xẻ. Trong môi trường phân tán, hy vọng mô phỏng được hệ thống bộ nhớ chia xẻ trong khi không có bộ nhớ vật lý chia xẻ. Khái niệm bộ nhớ phân tán này đưa ra một số câu hỏi về tính nhất quán và hiệu năng của chia xẻ dữ liệu là tương tự như chia xẻ file trong hệ thống file phân tán. Chương 6 trình bày về hệ thống file phân tán.

• Cuối cùng, song không kém quan trọng, là *quản trị file* trong HĐH. File là một thực thể dữ liệu lôgic được thi hành trên các thiết bị nhớ, bao gồm đĩa, bộ nhớ, và thậm chí cả thiết bị vào-ra. Theo nghĩa trừu tượng nhất, mọi tính toán được xem như các quá trình thao tác với file. Nếu bỏ đi hai thuật ngữ cơ bản là quá trình và file, thì không còn có gì nghiên cứu về HĐH. Do chúng ta chỉ giải quyết với quá trình và file, mọi chủ đề tiếp theo đều liên quan đến chúng. Chúng ta không bàn luận nhiều về quản trị vào - ra và quản trị bộ nhớ vì điều đó chỉ thích hợp trong HĐH tập trung.

File cần được cấu trúc và thi hành trước khi được thao tác. Mỗi khi một cấu trúc file chung và thi hành của chúng được quyết định thì các chức năng cơ sở để quản trị file là truy nhập file (file acces) và chia xẻ file. Thêm nữa vì mục tiêu hiệu quả, truy nhập file đòi hỏi cơ chế điều khiển bảo vệ (protection) và an toàn, và chia xẻ file đồng bộ hoặc điều khiển đồng thời. Khác với quản trị bộ nhớ và quản trị vào-ra, file được phân tán và nhân bản trên mạng hoặc môi trường phân tán. An toàn và điều khiển đồng thời file để thao tác file trở thành những vấn đề thiết thực hơn trong thiết kế HĐH phân tán so với HĐH tập trung. Ứng dụng caching trong truy nhập file cũng phức tạp hơn, do thực tế file được cache trên nhiều máy. Một số chương, đoạn tiếp thảo luận về thi hành và điều khiển hệ thống file phân tán.

1.3. Sơ lược về hệ điều hành mạng

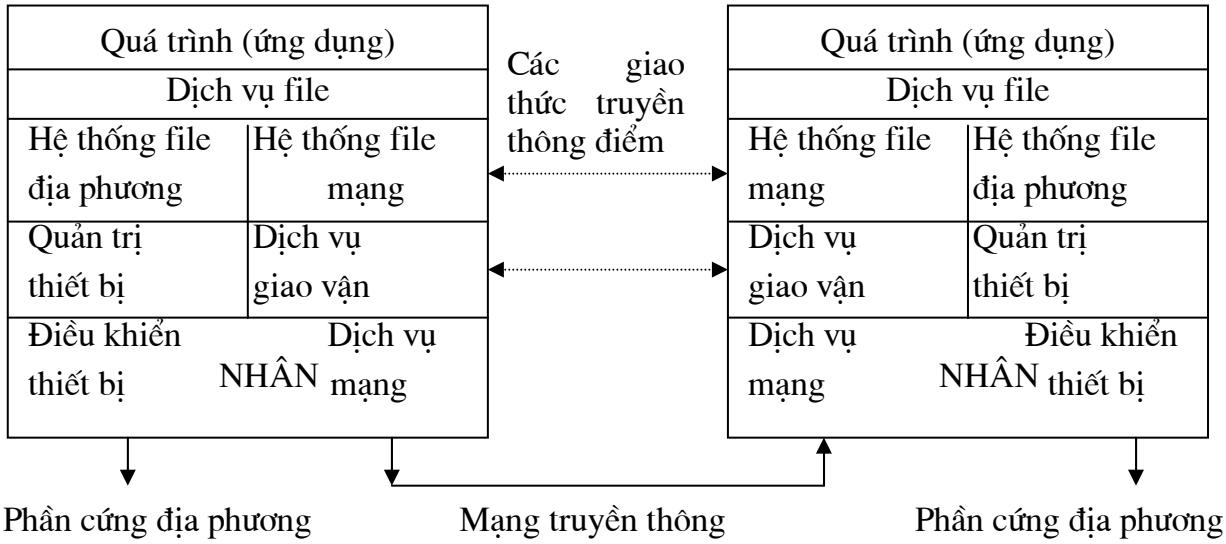
Hiện nay, máy tính không được dùng một cách riêng lẻ và nhiều máy tính được kết nối thành một hệ thống tính toán chung; mỗi máy tính đảm nhận một chức năng bộ phận và toàn bộ hệ thống tính toán chung đó có năng lực hơn hẳn việc sử dụng riêng lẻ. Theo tiến trình đó các loại HĐH mạng, HĐH phân tán và HĐH tự trị cộng tác xuất hiện (hình 1.3). So với HĐH tập trung, kết nối phần cứng và phần mềm trong hệ thống máy tính trở nên mềm dẻo hơn, trong một số trường hợp (như HĐH mạng) kết nối đó là lỏng lẻo.

HĐH mạng cho phép liên kết nhiều máy tính theo cách không thực sự chặt chẽ: không có sự điều khiển phần cứng hoặc phần mềm trực tiếp từ một trạm làm việc (workstation) tới những trạm làm việc khác tồn tại trong hệ thống, và tổng phí truyền thông giữa các trạm cuối (đo theo thời gian) là lớn hơn rất nhiều so với chuyển giao thông tin nội tại trong mỗi trạm cuối. Mục tiêu căn bản của HĐH mạng là chia xẻ tài nguyên (bao gồm chương trình và dữ liệu) trong mạng. Tương tác duy nhất trong hệ thống là trao đổi thông tin giữa các trạm thông qua một vài dạng kênh truyền thông ngoài. Đặc trưng duy nhất, liên thao tác (interoperability) là tính chất mong muốn trong hệ thống máy tính mạng. Liên thao tác cung cấp tính linh hoạt trong trao đổi thông tin dọc theo các trạm trong mạng máy tính hỗn tạp, đây được gọi là tính liên tác động. Liên thao tác được biểu thị bởi các giao thức truyền thông chuẩn và giao diện chung nhằm chia xẻ CSDL và hệ thống File. Ví dụ về cơ chế hỗ trợ liên thao tác là giao thức truyền thông chuẩn và giao diện chung tới các CSDL (data base) hoặc hệ thống file.

Chức năng trao đổi thông tin được phân chia và thi hành theo cấu trúc mức. Tại mức phần cứng, mạng con truyền thông chịu trách nhiệm thi hành trao đổi thông tin. Cao hơn, HĐH cung cấp dịch vụ giao vận (transport service) dữ liệu và người dùng sử dụng các giao thức truyền thông quá trình điểm - điểm (peer to peer) hướng ứng dụng. Các mức có thể mịn hơn như kiến trúc bảy mức OSI của ISO.

HĐH mạng có thể được coi là mở rộng trực tiếp HĐH truyền thống, được thiết kế nhằm làm thuận tiện chia xẻ tài nguyên và trao đổi thông tin. Do đó, thuận tiện mô tả HĐH mạng thông qua minh họa các ứng dụng mạng chung của nó và các dịch vụ giao vận cần có để hỗ trợ các ứng dụng này. Dịch vụ giao vận phục vụ như một giao diện đứng giữa QT ứng dụng mạng và mạng truyền thông vật lý, và nó thi hành giao thức truyền thông giữa hai hệ điều hành điểm. Hình 1.6 cho thấy sự tích hợp các dịch vụ giao vận trong HĐH đối với QT ứng dụng truy nhập hệ thống file từ xa. Ví dụ này được mô hình hóa theo Hệ thống file mạng (Network File System: NFS) của Sun. Truy nhập file từ xa dựa trên hệ thống file mạng truyền thông và được chuyển dịch bởi hệ thống mạng thành các giao vận dữ liệu giữa các dịch vụ điểm.

Hầu hết các HĐH mạng dùng API mức cao chẳng hạn như socket và lời gọi thủ tục từ xa (Remote Procedure Call: RPC) đối với dịch vụ giao vận nhằm hỗ trợ truyền thông giữa các HĐH trong các miền mạng khác nhau. HĐH mạng được đặc trưng bởi tập gồm một mức giao vận và hỗ trợ ứng dụng mạng chạy trên dịch vụ giao vận. Các lớp ứng dụng mạng đáng chú ý là đăng nhập từ xa (remote login), chuyển file (file transfer), thông điệp, duyệt mạng (network browsing) và thực hiện từ xa (remote execution). Dưới đây trình bày sơ lược về chúng.



Hình 1.6. Tích hợp dịch vụ giao vận

• Đăng nhập từ xa: là khả năng cho phép trạm riêng của người dùng thành một trạm cuối đăng nhập vào một trạm làm việc từ xa trong mạng, cho phép chia xẻ trực tiếp CPU và tài nguyên tương ứng của nó. Để đăng nhập từ xa input từ bàn phím được chuyển đổi thành các bó dữ liệu của các giao thức truyền thông mạng. Tại điểm đối ngẫu áp dụng tới hiển thị output. Đôi lúc hy vọng mô phỏng rất nhiều kiểu trạm cuối (được gọi là mô phỏng trạm cuối). Như vậy, việc dàn xếp giữa các tham số trạm cuối là cần thiết trước khi kết nối được thiết lập. Dịch vụ với mở rộng kết hợp này được gọi là hỗ trợ trạm cuối ảo. Một ứng dụng mạng được sử dụng rộng rãi với mở rộng như vậy là ***telnet***, một dịch vụ đăng nhập từ xa được thiết kế cho các trạm cuối không đồng bộ (asynchronous: dị bộ). Trong UNIX, ***rlogin*** là dịch vụ tương tự ngoại trừ nó không hỗ trợ mô phỏng trạm cuối. Thêm vào, rlogin giả thiết rằng host từ xa trong cùng một miền đồng nhất, và việc xác minh mật khẩu không phải là một lựa chọn ngầm định.

• *Truyền file*: là năng lực truyền file hoặc mang chuyển file dọc theo các trạm làm việc khác nhau trong một hệ thống mạng. Truyền file không đơn thuần một trao đổi dữ liệu. File chứa dữ liệu, cấu trúc file và cả các thuộc tính file. Như vậy, một giao thức truyền file (chẳng hạn, ***fpt*** trong UNIX) bắt buộc cung cấp một giao diện tới các hệ thống file địa phương và hỗ trợ các lệnh tương tác của người dùng. Thông tin về thuộc tính file, khuôn dạng dữ liệu, dòng dữ liệu, và điều khiển truy nhập bắt buộc phải được trao đổi và có giá trị như một phần của thao tác truyền file. Nhân bản file từ xa (***rcp***) trong UNIX là một dịch vụ truyền file có hạn chế bằng việc sao các fioe giữa các trạm làm việc, khi giả thiết rằng cấu trúc file UNIX là như nhau trong miền mạng (tức là HĐH tại các nút là đồng nhất).

• Hệ thống *thông điệp* cho phép người sử dụng mạng gửi và nhận tài liệu hoặc thông điệp mà không cần tạo ra một kết nối thời gian thực. Hai ứng dụng thông điệp chính là Chuyển đổi dữ liệu điện tử (Electronic Data Interchange: EDI) đối với các giao dịch (transaction) kinh doanh và thư điện tử (e-mail). EDI là ứng dụng chuẩn mà nguyên tắc chủ đạo là truyền thông tin kinh doanh. E-mail là thông điệp cho phép trao đổi thông điệp giữa các người dùng mạng. Khác với truyền File, hệ thống mail là không thông dịch ngoại trừ những thông điệp chung được gắn vào trong mail (hiện nay, điều này không hoàn toàn do một số hệ thống mail có chức năng thực hiện từ xa). Thuộc tính cấu trúc và điều khiển truy nhập của dữ liệu mail không được chú ý. Điều căn bản là nắm giữ và truyền thông điệp và giao diện người dùng thao tác trên thông điệp mail. Rất nhiều chuẩn, chẳng hạn X.400 do CCITT (nay là, ITU-T) và Giao thức truyền mail

đơn giản (Simple Mail Transfer Protocol: SMTP) của Bộ quốc phòng Mỹ, đã được đề xuất nhằm thi hành hệ thống mail mạng. Nhiều hệ thống e-mail tinh vi đã được xây dựng, để phục vụ như bộ chuyển đổi có năng lực truyền thông giữa các hệ thống mail khác nhau.

• *Duyệt mạng* là dịch vụ thông tin để tìm kiếm và trình bày các tài liệu giữa các site mạng thành viên. Trình duyệt thường được thi hành như là một hệ thống Client/Server trong đó trình duyệt là khách truy nhập đối tượng tại phục vụ file từ xa. Hệ thống được sử dụng rộng rãi nhất hiện nay là WWW (World Wide Web). WWW là mô hình dữ liệu để liên kết các tài liệu siêu phương tiện dùng các chỉ dẫn được gọi là Bộ định vị tài nguyên thống nhất (Universal Resource Locator: URL). Tài liệu được hiển thị bởi trình duyệt thường là siêu văn bản (hypertext) và có thể chứa nhiều con trỏ tới siêu văn bản khác hoặc siêu phương tiện khác. Trình duyệt, chăng hạn Mosaic, truyền thông với phục vụ WWW dùng giao thức truyền siêu văn bản (HyperText Transport Protocol: HTTP). Các giao thức khác, chẳng hạn ftp và telnet cũng được sử dụng. Tài liệu đa phương tiện điển hình được cấu trúc khi sử dụng Ngôn ngữ đánh dấu văn bản (HyperText Markup Language: HTML) và được phân tán nhờ dịch vụ Web. Hiện có nhiều hệ thóng duyệt khác với (cơ sở dữ liệu) tài nguyên phân tán lớn. Vào thời điểm 1997, Netscape hầu như là hệ thống duyệt phổ dụng nhất với hiệu quả bổ sung của nó và sự mở rộng về an toàn.

• *Thực hiện từ xa* là khả năng gửi thông điệp đòi hỏi sự thực hiện một chương trình tại site từ xa. Do các chương trình thực hiện được là phụ thuộc máy và không thể chạy trên máy tùy ý, sự thực hiện từ xa thường được làm theo cách thông dịch (không là biên dịch) một file script hoặc mã liên phương tiên độc lập máy được thông điệp đưa ra. Thực hiện từ xa là một công cụ mạng rất mạnh song nguy hiểm. Vì thế nó thường được giới hạn tới một số ứng dụng mà sự hạn chế có thể kéo theo việc ngăn ngừa đe doạ và bảo vệ khỏi vi phạm.

Ứng dụng tốt của thực hiện từ xa là chuyển vận dữ liệu đa phương tiện. File video và ảnh đòi hỏi khối lượng lớn băng thông nếu chúng được truyền dưới dạng dòng điểm. Chúng cũng phải gặp bài toán về tính không tương thích trong hiển thị output. Một số ngôn ngữ liên phương tiện phổ dụng có thể được dùng để đặc tả dạng thống nhất và cô đọng hơn. Tại điểm nhận, thông dịch tương ứng được gọi nhằm dịch dữ liệu hoặc thực hiện các chỉ thị trong ngôn ngữ đa phương tiện. Vấn đề chuyển đổi dữ liệu được giải quyết và việc tải trên mạng là rất lớn.

Nhiều ứng dụng mạng sử dụng khái niệm thực hiện từ xa. Ví dụ, MIME (Multupurpose Internet Mail Extension) là hệ thống mail tích cực mà hỗ trợ trao đổi mail đa phương tiện giữa các máy tính khác nhau, chẳng hạn, thông điệp có thể mang một kiểu đặc biệt cho một hiển thị riêng. Phụ thuộc vào kiểu, quá trình tương ứng được gọi nhằm thực hiện bài toán. Thông điệp ytong mail MIME được thông dịch và có cùng hiệu quả như chương trình thực hiện ở xa.

Cách tiếp cận tổng quát hơn tới thực hiện từ xa trong ngôn ngữ và môi trường lập trình Java. Java là ngôn ngữ lập trình hướng đối tượng mục đích - tổng quát, xuất pát từ C++. Biên dịch Java cho dãy các chỉ thị mã-byte hiệu năng cao và độc lập máy cô đọng có thể được gửi và thông dịch tại host bất kỳ miễn có sẵn thông dịch Java. Chương trình mã-byte được gọi là tiểu dụng (applet). Nhằm hỗ trợ dịch vụ mạng và phân tán, môi trường lập trình Java cung cấp thư viện gồm các thủ tục con kết hợp chặt chẽ các giao thức Internet, chẳng hạn http và fpt. Một tiểu dụng Java là một đối tượng mà có thể được chỉ dẫn tại một URRL nhằm mở các đối tượng khác. Một ứng dụng trực tiếp của tiểu dụng trong WWW là sử dụng thế mạnh động của tiểu dụng để kéo

ảnh được tạo ra dễ dàng hơn trong hệ thống duyệt. Phiên bản mới của Netscape được thi hành nhờ sử dụng Java vì vậy hỗ trợ tiểu dụng Java.

Do việc sử dụng những ứng dụng chia sẻ tài nguyên mạng như trên đang phát triển, chúng được thi hành như những phục vụ hệ thống chuẩn (quá trình chạy ngầm: daemon) thực hiện giao thức điểm trên một hạ tầng dịch vụ giao vận và trở thành bộ phận của HĐH mạng.

• Ngoài ra, hiện có nhiều hướng cải tiến truyền thông trên mạng liên quan đến tính chất lin hoạt, thường được gọi là "tích cực" trong mạng, liên quan đến giao thức (thông điệp tích cực: active massage), liên quan đến môi trường (mạng tích cực: active network) … Mặt khác, an ninh mạng đã và đang là một trong những vấn đề cốt lõi nhất hiện nay.

1.4. Sơ lược về hệ điều hành phân tán

HĐH phân tán mới thực sự là một HĐH quản lý tài nguyên máy tính trên phạm vi lãnh thổ lớn. Các máy tính được kết nối lôgic (theo phần mềm) trong HĐH phân tán một cách tương đối chặt chẽ, hệ thống tài nguyên của mỗi máy tính đóng góp thực sự vào hệ thống tài nguyên chung thống nhất và tham gia vào việc giải quyết mỗi bài toán điều phối quá trình, điều phối bộ nhớ, điều phối vào-ra v.v. HĐH phân tán, về lôgic là một hệ thống thống nhất song về vật lý lại được "phân bố" chạy trên nhiều máy tính ở các vị trí khác nhau.

Sự phát triển các trạm làm việc mạnh và những tiến bộ của công nghệ truyền thông tạo ra sự cần thiết và hợp lý để mở rộng việc chia xẻ tài nguyên thêm một bước nữa: để bao gồm dạng tổng quát hơn nữa các hoạt động cộng tác giữa một tập hợp gồm các máy tính tự trị, được kết nối bởi một mạng truyền thông. Chia xẻ tài nguyên và cộng tác các hoạt động phân tán kiểu này của môi trường tính toán là những mục tiêu chính trong thiết kế HĐH phân tán và là tiêu điểm chính của tập bài giảng này.

Cần xác định những thành phần trong một hệ phân tán kết nối lỏng là cần phân tán hay không tập trung. *Tài nguyên vật lý* là phân tán vì được thừa hưởng tự nhiên từ hệ kết nối lỏng. *Thông tin và nhu cầu thông tin* trở nên phân tán do tính tự nhiên của nó hoặc do nhu cầu tổ chức, chẳng hạn về tính hiệu quả và tính an toàn. Hơn nữa, hiệu năng hệ thống cần được nâng cao nhờ *tính toán phân tán*. Làm thế nào để các tài nguyên và hoạt động phân tán được quản lý và điều khiển là những trách nhiệm căn bản của HĐH phân tán. Nên chăng HĐH phân tán tự nó cũng phân tán ? Lời giải đáp là về đại thể là nên theo cách thức đó chính do tính tự nhien của nó và nhu cầu tổ chức. Điều đó đặt ra vấn đề thi hành phân tán đối với các chức năng quản trị và điều khiển của HĐH phân tán, chính là thiết kế các thuật toán phân tán. Nhu cầu về các thuật toán phân tán trong các HĐH phân tán thúc đẩy việc tích hợp hai chủ thể có quan hệ mật thiết này trong một số tài liệu.

Khi cho một HĐH phân tán trên một hệ phân tán, hy vọng có được sự che khuất các chi tiết thi hành của hệ thống đó đối với người dùng. Điều phân biệt mấu chốt giữa HĐH mạng và HĐH phân tán ở chính khái niệm *trong suốt*. Trong suốt là một khái niệm mới. Trong HĐH tập trung, người sử dụng chia xẻ thời gian có sự *trong suốt đồng thời* (concurrency transparency) nếu họ không nhận biết thực tế có nhiều người dung fkhác cũng đang chia xẻ cùng một hệ thống. Một chương trình được gọi *trong suốt định vị* (location transparrency) nếu như bản đồ của chương trình vào trong bộ nhớ vật lý và/hoặc bộ xử lý là bị che khuất. Trong HĐH phân tán khái niệm này còn được mở rộng tới định vị file và đồng thời truy nhập nếu file có thể nằm bất kỳ trên hệ thống lưu trữ và truy nhập nó thông qua đường dẫn lôgic hơn là vật lý. Đối với quá

trình phân tán, có thể đạt được phân tán song song và phân tán hiệu năng nếu quá trình có thể được thực hiện trên một bộ xử lý bất kỳ mà không kể sự nhận biết của người dùng và không kể sự khác nhau đáng kể về hiệu năng. Còn nhiều ví dụ nữa và có giới hạn hay không ? Một hệ thống trong suốt hoàn toàn là hợp lý hoặc thậm chí chỉ hy vọng là một câu hỏi còn được bàn luận. Nói chung, tính trong suốt là một cái tốt đẹp cần có và chúng ta vẫn sử dụng nó như mở rộng mấu chốt của HĐH phân tán.

Trong các mục trước đây, hệ thống tính toán được mô tả như một hệ thống trừu tượng bao gồm các quá trình và các file. Cần bổ sung các thuật toán (chính xác hơn là các thuật toán điều khiển phân tán) mà quản lý sự thực hiện các quá trình trên các file trong hệ phân tán. Như vậy, HĐH phân tán bao gồm ba thành phần chính: điều phối các quá trình phân tán, quản trị các tài nguyên phân tán và thi hành các thuật toán phân tán. Tại mỗi nút trong hệ phân tán, giả thiết rằng tồn tại những môđun thực hiện việc quản trị tài nguyên địa phương.

Một số HĐH phân tán điển hình như AMAEBA, MACH, CHORUS, DCE được giới thiệu trong [8].

1.5. Sơ lược về hệ tự trị cộng tác

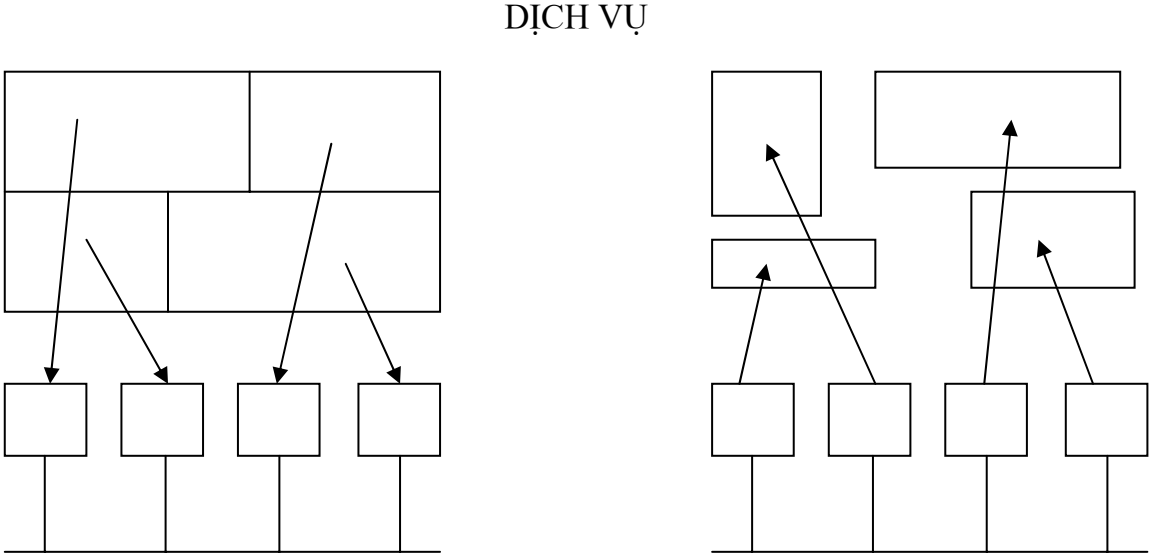
HĐH tự trị cộng tác cho một cách thức linh hoạt hơn so với HĐH phân tán. Các máy tính thành viên vừa được phép tham gia kết nối vào toàn bộ hệ thống lại vừa được phép chạy một cách độc lập. Khi tham gia vào hệ thống, tài nguyên của máy tính thành viên được toàn bộ hệ thống sử dụng (gần như theo cách thức của HĐH phân tán) còn khi máy thành viên chạy độc lập thì nó độc quyền sử dụng tài nguyên riêng. Về thực chất, trong hệ tự trị cộng tác, tính "tự trị" của máy thành viên được chú trọng hơn so với tính thống nhất lôgic của toàn bộ hệ thống.

Như vậy, nếu chỉ cần duy trì tính trong suốt ở một mức độ nào đó và hủy bỏ về cái nhìn của một hệ thống nhất lôgic của hệ đa máy tính, nhận được cách nhìn khác nhau hoàn toàn của một hệ (phần cứng và phần mềm) lỏng lẻo thuần túy. Mỗi người dùng hoặc quá trình thao tác tự trị bằng cách cung cấp các dịch vụ của mình và yêu cầu các dịch vụ từ nơi khác. Nhóm các hành động có thể được điều phối bằng việc trao đổi dịch vụ và yêu cầu. Dịch vụ mức cao có thể được cung cấp bằng cách giải quyết chúng từ những dịch vụ ở mức thấp hơn. Mọi hệ thống phần mềm có thể được định danh một cách thoải mái bằng cách tích hợp nhiều dịch vụ với sự thoả thuận nào đó theo cấu trúc. Đây là cách tiệm cận đã bắt chước cách ứng xử trong xã hội loài người: Ứng xử trong hệ thống máy tính làm theo cách ứng xử trong xã hội loài người phức tạp. Đây là cách nhìn của hệ tự trị cộng tác. Hình 1.8 minh họa một số khác biệt cơ bản giữa HĐH phân tán với hệ tự trị công tác. Hệ phân tán được đặc trưng bằng phân tích dịch vụ trong khi hệ tự trị cộng tác lại nhấn mạnh việc tích hợp dịch vụ.

Hệ tự trị cộng tác là hệ thống phần mềm định hướng dịch vụ mức cao đòi hỏi hỗ trợ cơ chế truyền thông trên đó các giao thức truyền thông mức cai đã được xây dựng. Lấy ví dụ hình ảnh cách thức giao dịch bất động sản có thể được thực hiện trong một hệ tự trị cộng tác. Người mua nhà, là một quá trình khách, có thể tạo ra một yêu cầu tới hoặc trực tiếp tới chủ ngôi nhà hoặc gián tiếp tới đại lý bất động sản (cả hai đều là quá trình phục vụ). Chủ ngôi nhà là quá trình khách tới người môi giới. Người môi giới có thể từ một đại lý bất động sản, một phục vụ lớn hơn có thể chỉ dẫn cho người mua nhà một môi giới giành riêng. Người bán là khách tới đại lý bất động sản giống như một khách tới người môi giới. Người mua có thể định vị được đại lý bất động sản nhờ xem thông tin trên Trang vàng, đã được biết đến như một quá trình phục vụ trực tiếp. Nếu ngôi nhà được chủ của nó bán trực tiếp thì người chủ có thể quảng cáo tại đâu đó nhờ quá trình phục vụ với địa chỉ đã biết. Hình 1.9 trình bày một loạt các quan hệ Client/Server

của một ứng dụng hệ tự trị cộng tác. Phục vụ kiểu Trang vàng và đại lý bất động sản cung cấp dịch vụ môi giới hoặc thương mại như những dịch vụ định vị. Khái niệm mấu chốt của hệ tự trị cộng tác là tích hợp các dịch vụ thành dạng hoạt động cộng tác. Cả phần cứng và phàn mềm là tách rời và không tập trung hoàn toàn.

Tư tưởng của hệ tự trị không tập trung hình như gượng gạo. Tuy nhiên, một mở rộng đơn giản có khái niệm Làm việc cộng tác được hỗ trợ bằng máy tính (Computer Supported Cooperative Work: CSCW). CSCW là một khung nhằm hỗ trợ phần mềm nhóm (groupware), một ứng dụng phần mềm lớn mà bao gồm các người dùng cộng tác và tài nguyên phân tán dọc theo một mạng hỗn tạp. Một ví dụ là hội thảo phân tán, trong đó cuộc mit tinh điện tử trong một mạng vật lý phân tán có thể được tổ chức. Trái ngược với triết lý máy tính đơn được chỉ cho người dùng và tài nguyên có thể thiết kế

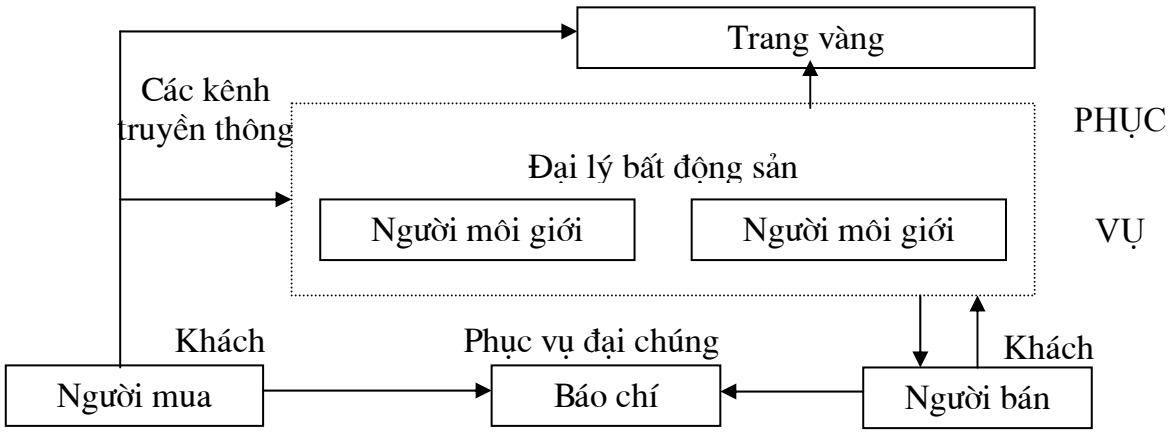


Hình 1.8. Phân tích và tích hợp dịch vụ

và quản trị, là sự nhận biết rõ ràng sự tồn tại đa máy tính. Người dùng, từ mạng logic của họ với mụctiêu riêng và được sẵn sàng cho điều khiển truy nhập và bảo vệ của nhóm. Hệ cộng tác không tập trung là hệ thống cungg cấp những dịch vụ chuẩn cho phép tích hợp các dịch vụ cộng tác mức cao trong một hệ thống mạng lớn. Với số lượng rất lớn đã lên phương án về mạng và con người, đây là sự tiến hóa tự nhiên của HĐH mạng và HĐH phân tán.

Nhu cầu trộn các ứng dụng hệ tự trị cộng tác có thể bùng phát một số cố gắng chuẩn hóa cho việc phát triển tương lai của phần mềm phân tán, đáng chú ý là Quá trình phân tán mở (Open Distributed Processing: ODP) và Kiến trúc môi giới yêu cầu đối tượng chung (Common Object Request Broker Architeturre: CORBA). ODP là khung hệ thống công cộng hõ trợ phân tán, liên thao tác và khả chuyển đối với các xử lý phân tán hỗn tạp cả bên trong và dọc theo tổ chức tự trị. CORBA ccung cho cùng triết lý và sử dụng mô hình hướng đối tượng để thi hành yêu cầu dịch vụ trong suốt dọc theo một hệ thống phân tán đa đối tượng hỗn tạp liên kết nối. Cả ODP và CORBA dùng dịch vụ thông minh ***trader*** hoặc ***broker*** làm thuận tiện liên tương tác trong hệ tự trị cọng tác. Trang vàng và đại lý bất động sản như những thương nhân. Chúng có thể được nhìn

như tuyến phần mềm liên kết quá trình khách và phục vụ và chúng phục vụ như một phần mềm lớp giữa (middleware) hỗ trợ các ứng dụng cộng tác phân tán.



Hình 1.9. Một hệ tự trị thông qua tích hợp dịch vụ

• Đồng thời với tiến trình phát triển trên đây của các HĐH, việc nghiên cứu về các *hệ thống xử lý song song* cũng được phát triển. Tương ứng với các mô hình song song trên các hệ thống tập trung SIMD, MISD, MIMD là các mô hình SPMD (Single Program Multiple Data), MPSD, MPSD trong đó đối tượng thực hiện song song là chương trình thay cho chỉ thị (instruction).

Một hướng nghiên cứu thời sự hiện nay là mô hình tính toán cụm (Cluster Computing) trong đó việc song song hóa một cách hiệu quả là mục tiêu của các mô hình như vậy. Trong các mô hình tính toán song song thì cách thức SPMD là điển hình nhất.

Tính toán song song trên mạng các máy tính cá nhân, khai thác công suất dư thừa của các máy tính cá nhân trong mạng cũng là hướng đang được đặc biệt chú ý, theo đó tìm cách "tổ hợp sức mạnh" các máy tính cá nhân trong mạng thành "siêu máy tính ảo" (có người còn gọi là "siêu máy tính con nhà nghèo). Cách thức nói trên liên quan đến việc tạo dựng "cụm máy tính cá nhân" (PC-cluster) bằng một hệ thống phần mềm (thuộc dạng middleware) với tên gọi là phần mềm PC-cluster. Hiện tại có hai lớp phần mềm PC-cluster miễn phí điển hình là PVM (Parallel Vitural Machine) và MPI (Message Passing Interface). Tính đến thời điểm năm 2002, một số hệ thống PC-cluster đã được cài đặt thử nghiệm tại một số cơ quan trong nước (trong đó có khoa Công nghệ, ĐHQGHN) song hiệu quả thực sự của chúng hiện vẫn còn ở mức rất khiêm tốn.

Vấn đề thiết kế và nghiên cứu đối với HĐH tập trung (truyền thống), hoạt động trong một hệ thống có một hoặc nhiều bộ xử lý, đã được nghiên cứu tương đối dầy đủ. Tuy nhiên, với việc phát triển nhanh chóng các trạm làm việc cá nhân và mạng cục bộ dẫn đến sự phát triển nhanh chóng các khái niệm HĐH mới, là HĐH mạng và HĐH phân tán (một số tác giả, đặc biệt là các tác giả Việt kiều, dùng thuật ngữ "phân bố" thay cho thuật ngữ "phân tán" được dùng trong tài liệu này). Vấn đề quan hệ đến mạng và HĐH phân tán là mục tiêu nghiên cứu của giáo trình này. Một vấn đề khác nổi lên là phát triển các hệ thống tự động cộng tác, trong đó nhấn mạnh việc thiết kế các thuật toán phân tán trong một môi trường hệ thống mở. Một hệ thống mở liên quan đến tính mềm dẻo một cách toàn vẹn và che khuất đi sự hỗn tạp các thành phần nhằm hỗ trợ việc cộng tác nhiều cấp tại mức ứng dụng. Khái niệm này là rộng lớn hơn so với HĐH theo nghĩa truyền thống.

1.6. Thuật toán phân tán

Việc thiết kế các thuật toán phân tán, được đòi hỏi nhằm hỗ trợ việc thi hành dịch vụ HĐH phân tán để điều phối sự thực hiện của các quá trình đồng thời có vai trò quan trọng trong nghiên cứu về HĐH phân tán. Các thuật toán thường được chỉ dẫn như các giao thức do chức năng của chúng là chủ yếu thiết lập lệnh hoặc quy tắc đối với sự hạn chế của hệ phân tán là thiếu những thông tin trạng thái hệ thống toàn cục. Mỗi quá trình có nhận thức khác nhau của hệ thống do sự thiếu vắng bộ nhớ chia xẻ và độ trễ truyền thông đáng kế giữa các quá trình. Cái nhien của họ về hệ thống thường là không đầy đủ và không mạch lạc. Phần tử bản chất nhất của thông tin toàn cục là thông tin thời gian toàn cục cỡ hệ thống, thường được chỉ dẫn như một đồng hồ toàn cục. Về mặt lý thuyết, không thể đạt được nhằm đạt được một đồng hồ toàn cục thậm chí trong hệ phân tán có một đồng hồ trung tâm chung. Bỏ qua thông tin thời gian toàn cục, sự thúc ép sự sắp xếp sự xuất hiện các sự kiện trở thành một bài toán không tầm thường. Việc xấp xỉ đồng hồ toàn cục với sự thứ lỗi thời gian nào đó và cơ chế thực hiện thứ tự sự kiện đúng đắn không cần sử dụng thông tin đồng hồ toàn cục bắt buộc phải được phát triển. Nhiều thi hành của các chức năng điều khiển mức cao chẳng hạn ĐBQT và TTQT dựa vào sự thứ lỗi (fault tolerance) thời gian và cơ chế sắp xếp sự kiện này. Độ trễ truyền thông tạo ra khó khăn lớn để đạt được thỏa thuận về trạng thái hệ thống, bản chất của các hoạt động phân tán cộng tác.

Bổ sung tới tính phức tạp do độ trễ truyền thông, thiết kế thuật toán phân tán là phức tạp hơn bởi vì nguồn lỗi và không tin cậy là phổ biến hơn trong hệ phân tán so với hệ tập trung. Thứ lỗi trong hệ phân tán là vấn đề khó tính hơn đối với các thuật toán phân tán. Bản chất là hệ thống bao gói nhiều kiểu của lỗi. Thậm chí nhiều thuật toán tập trung để ĐBQT, lập lịch, và điều khiển đồng thời .. buộc phải được xem xét kỹ lưỡng để dùng trong hệ phân tán. Thuật toán có thể được phân thành hai lớp: thuật toán không tập trung đầy đủ và thuật toán phân tán với một điều phối tập trung thứ lỗi. Loại thứ hai đơn giản hơn theo khái niệm cung cấp những cơ chế hiệu quả tồn tại nhằm kiểm soát lỗi của điều khiển tập trung và chọn những chỉ đạo mới.

Kiến trúc phần cứng của hệ phân tán cũng có vai trò quan trọng trong thi hành các thuật toán phân tán. Các phương pháp truyền thông phụ thuộc vào việc tôpô mạng là kết nối đầy đủ hay không, thông thường hay không thông thường, và truyền dữ liệulà điểm-điểm hay đa điểm. Kiến trúc thậm chí cho phép cả việc thay đổi tôpô, lỗi kết nối và các nút là tồn tại. Về phía phần mềm dữ liệu thường được nhân bản nhằm cho phép truy nhập đồng thời và đạt được độ tin cậy cao hơn. Nhân bản dữ liêu lại đưa đến vấn đề tính chặt chẽ của dữ liệu. Quản lý nhân bản dữ liệu trở thành một vấn đề cũng khó tính trong thiết kế hệ phân tán.

Dưới đây là một danh sách tổng quát các thuật toán phân tán khi lưu tâm tới những vấn đề đáng kể của hệ phân tán được tóm tắt từ những điều mô tả trên.

• *Chuyển thông điệp* Hệ quả của việc không có bộ nhớ chia xẻ ngụ ý rằng điều phối giữa các quá trình đồng thời bắt buộc phải thực hiện bằng CTĐ. Như vậy, thuật toán đồng bộ và nắm giữ bế tắc cần được thiết kế lại trong môi trường phân tán. Thuật toán phân tán có thể không tập trung hoàn toàn hoặc tập trung. Trong thuật toán tập trung, thuật toán bầu cử phân tán thường được đòi hỏi để thiết lập và duy trì điều khiển tập trung.

• *Sự thiếu thông tin toàn cục*. Hiệu lực của thuật toán phân tán phụ thuộc vào tri thức của nó về trạng thái của hệ thống. Do không hợp lý nếu đưa ra thông tin trạng thái toàn cục do độ trễ mạng và các thành phần trong hệ thống không tin cậy, tương tác giữa các quá trình bắt buộc phải dựa trên sự nhất trí nhận được từ một vài giao thức thỏa thuận nào đó. Giao thức thoả thuận tự nó là thuật toán phân tán.

• *Nhân bản dữ liệu*. Quản lý nhân bản dữ liệu là chức năng cơ sở của hệ thống file và cơ sở dữ liệu phân tán. Mục tiêu căn bản của giao thức là duy trì tính nhất quán (consistency). Vấn đề tương đương lôgic cần đến tán phát tin cậy (reliable broadcast). Tập các dữ liệu được nhân bản là tương tự như một nhóm thành viên được tán phát. Vấn đề này kéo theo trong HĐH hoặc CSDL cũng được nhìn nhận.

• *Lỗi và khôi phục*. Độ tin cậy của hệ thống có thể được nâng cao theo nghĩa thứ lỗi hoặc khôi phục tiếp sau lỗi. Tiếp cận thứ lỗi sử dụng giải pháp dư thừa hoặc đa phục vụ. Khôi phục là cách tiếp cận sẵn có trong đó trạng thái của hệ thống là được duy trì và được dùng để thực hiện lại từ điểm kiểm tra ngay trước. Thuật toán khôi phục giải quyết với việc đăng nhập vào trạng thái hệ thống, các điểm kiểm tra và nắm giữ các quá trình và thông điệp cô lập.

CÂU HỎI VÀ BÀI TẬP

1. Trình bày khái niệm và hai chức năng cơ bản của hệ điều hành.
2. Trình bày sơ lược về quá trình tiến hóa của hệ điều hành, những nét đặc trưng nhất của mỗi lớp hệ điều hành. Nhận xét về quá trình tiến hóa đó.
3. Trình bày những bài toán điều khiển chủ yếu nhất của hệ điều hành truyền thống và sơ bộ về một số giải pháp giải quyết mỗi bài toán đó.
4. Khái niệm vi nhân và sơ bộ về giải pháp vi nhân.
5. Tính mở và tính khả chuyển của hệ điều hành. Sơ bộ về giải pháp thi hành tính mở và tính khả chuyển.

CHƯƠNG II. KHÁI NIỆM VÀ KIẾN TRÚC HỆ PHÂN TÁN

II.0. Giới thiệu

Như đã được trình bày trong chương trước, HĐH hiện đại thường tập trung vào chức năng máy tính ảo, nhấn mạnh mức dịch vụ hệ thống và vì vậy thuận tiện hơn quan niệm HĐH phân tán như một bộ tích hợp các dịch vụ hệ thống cho phép trình diễn cái nhìn trong suốt tới hệ thống máy tính với tài nguyên và điều khiển phân tán (đặt tại nhiều vị trí địa lý khác nhau). Có thể nói HĐH phân tán là HĐH *kết nối chặt về phần mềm* trên nền tảng *kết nối lỏng về phần cứng*. Theo một cách nói khác, HĐH phân tán cung cấp cho người sử dụng cách thức làm việc như với một HĐH tập trung trong điều kiện phân tán cả phần cứng lẫn phần mềm.

Một vấn đề đặt ra cho chính khái niệm HĐH phân tán. Tồn tại nhiều cách hiểu về HĐH phân tán, song có rất hiếm tài liệu cho một định nghĩa chính thức về HĐH phân tán. Trong nhiều ngữ cảnh, người ta còn sử dụng khái niệm "hệ phân tán" thay thế cho khái niệm "HĐH phân tán". Chúng ta chấp nhận định nghĩa được đưa ra trong [8]:

Hệ phân tán là tổ hợp bao gồm các máy tính độc lập với trình diễn hệ thống như một máy tính đơn trước người dùng.

HĐH phân tán được phát triển trên cơ sở một số tiền đề sau đây:

• Thứ nhất, do nhu cầu tăng không ngừng việc chia xẻ tài nguyên và thông tin mà các HĐH đã có từ trước không đáp ứng được.

Trong quá trình triển khai ứng dụng Tin học vào đời sống, các mạng máy tính được phát triển không ngừng, các tài nguyên của các máy tính trong mạng (phần cứng, phần mềm) ngày càng được mở rộng và nâng cấp, giá trị các tài nguyên này càng tăng nhanh dẫn đến sự tăng trưởng vượt bậc nhu cầu chia xẻ tài nguyên và thông tin trong một hệ thống thống nhất. HĐH tập trung và HĐH mạng thuần túy không đáp ứng được nhu cầu đối với sự tăng trưởng đó.

• Tiền đề thứ hai liên quan đến việc giá các trạm làm việc giảm nhanh chóng.

Việc giảm giá các trạm làm việc làm cho chúng được sử dụng phổ dụng hơn, số lượng và chất lượng các trạm làm việc cũng tăng không ngừng mà từ đó làm tăng yêu cầu xử lý phân tán. Điều này tạo ra nhiều vị trí có khả năng xử lý và lưu trữ thông tin hơn mà từ đó cần thiết phải phối hợp để chia xẻ tốt hơn tiềm năng lưu trữ và xử lý của các vị trí đó.

• Việc sử dụng rộng rãi các mạng

Trên cơ sở việc kết nối mạng để triển khai HĐH mạng tạo nên một cơ sở kỹ thuật hạ tầng (phần cứng, kết nối mạng, phần mềm) làm nền tảng phát triển HĐH phân tán.

• Tính thuần thục về kỹ nghệ phần mềm của các chuyên gia phát triển HĐH. Kinh nghiệm xây dựng HĐH trước đây (HĐH tập trung, HĐH mạng) cho phép nâng cao trình độ để đủ năng lực xây dựng HĐH phân tán.

II.1. Các mục tiêu thiết kế hệ điều hành phân tán

I.1.1. Đặc điểm của hệ phân tán

Hệ phân tán có các đặc điểm cơ bản là *Tính chia xẻ tài nguyên*, *Tính mở*, *Khả năng song song*, *Tính mở rộng*, *Khả năng thứ lỗi*, *Tính trong suốt*.

a. Tính chia xẻ tài nguyên

Thuật ngữ tài nguyên được dùng để chỉ tất cả mọi thứ có thể được chia xẻ trong hệ phân tán, bao gồm từ các thiết bị phần cứng (Đĩa, máy in ...) tới các đối tượng (file, các cửa sổ, CSDL và các đối tượng dữ liệu khác).

Trong hệ phân tán, chia xẻ tài nguyên được hiểu là tài nguyên của hệ thống được các QT chia xẻ (sử dụng chung) mà không bị hạn chế bởi tình trạng phân tán tài nguyên theo vị trí địa lý.

Việc chia xẻ tài nguyên trên hệ phân tán - trong đó tài nguyên bị lệ thuộc về mặt vật lý với một máy tính nào đó - được thực hiện thông qua truyền thông. Để chia xẻ tài nguyên một cách hiệu quả thì *mỗi tài nguyên cần phải được quản lý* bởi một chương trình có giao diện truyền thông, *các tài nguyên có thể truy nhập, cập nhật được* một cách tin cậy và nhất quán. Quản lý tài nguyên ở đây bao gồm lập kế hoạch và dự phòng, đặt tên các lớp tài nguyên, cho phép tài nguyên được truy cập từ nơi khác, ánh xạ tên tài nguyên vào địa chỉ truyền thông ...

b. Tính mở

Tính mở của một hệ thống máy tính là *tính dễ dàng mở rộng phần cứng* (thiết bị ngoại vi, bộ nhớ, các giao diện truyền thông ...) *và phần mềm* (các mô hình HĐH, các giao thức truyền thông, các dịch vụ chia xẻ tài nguyên ...) của nó. Nói một cách khác, tính mở của hệ thống phân tán mang ý nghĩa bao hàm tính dễ dàng cấu hình cả phần cứng lẫn phần mềm của nó.

Tính mở của hệ phân tán được thể hiện là hệ thống có thể *được tạo nên từ nhiều loại phần cứng và phần mềm* của nhiều nhà cung cấp khác nhau với điều kiện các thành phần này phải *theo một tiêu chuẩn chung* (liên quan đến HĐH là *tính đa dạng tài nguyên*; liên quan đến nhà cung cấp tài nguyên là *tính chuẩn*). Vai trò của ASP và SPI trong HĐH đã được trình bày trong chương 1.

Tính mở của Hệ phân tán được xem xét theo mức độ *bổ sung thêm các dịch vụ chia xẻ tài nguyên mà không phá hỏng hay nhân đôi các dịch vụ* đang tồn tại. Tính mở được hoàn thiện bằng cách xác định hay phân định rõ các giao diện chính của hệ phân tán và làm cho nó tương thích với các nhà phát triển phần mềm (tức là các giao diện chính của HĐH phân tán cần phổ dụng).

Tính mở của HĐH phân tán được thi hành dựa trên việc cung cấp *cơ chế truyền thông giữa các QT và công khai các giao diện được dùng để truy cập tài nguyên chung*.

c. Khả năng song song

Hệ phân tán hoạt động trên một mạng truyền thông có nhiều máy tính, mỗi máy tính có thể có một hoặc nhiều CPU. Trong cùng một thời điểm nếu có từ hai QT trở lên cùng tồn tại, ta nói rằng chúng được thực hiện đồng thời. Việc thực hiện các QT đồng thời theo cơ chế phân chia thời gian (một CPU) hay song song (nhiều CPU).

Khả năng làm việc song song trong hệ phân tán được thi hành do hai tình huống:

- Nhiều người sử dụng đồng thời đưa ra các lệnh hay tương tác với chương trình ứng dụng (đồng thời xuất hiện nhiều QT khách).

- Nhiều QT phục vụ chạy đồng thời, mỗi QT đáp ứng yêu cầu của một trong số các QT Khách.

Từ điều kiện đa xử lý, khả năng song song của hệ thống phân tán trở thành một thuộc tính của nó.

d. Khả năng mở rộng

Hệ phân tán có khả năng hoạt động tốt và hiệu quả ở nhiều mức khác nhau. Một hệ phân tán nhỏ nhất có thể hoạt động chỉ cần hai trạm làm việc và một phục vụ file. Các hệ lớn có thể bao gồm hàng nghìn máy tính, nhiều phục vụ File và phục vụ máy in ...

Khả năng mở rộng của một hệ phân tán được đặc trưng bởi *tính không thay đổi phần mềm hệ thống và phần mềm ứng dụng khi hệ thống được mở rộng*.

Điều này chỉ đạt ở mức độ nào đó đối với hệ phân tán hiện tại (không thể hoàn toàn như định nghĩa trên). Yêu cầu mở rộng không chỉ là mở rộng về phần cứng hay về mạng trên đó hệ thống bao trùm mà còn cần phải được phân tích, đánh giá trên tất cả các khía cạnh khi thiết kế hệ phân tán. Một ví dụ đơn giản là tình huống tần suất sử dụng một file quá cao xuất hiện như kết quả của việc tăng số người sử dụng trên mạng. Để tránh tình trạng tắc nghẽn xảy ra nếu như chỉ có một phục vụ đáp ứng các yêu cầu truy cập file đó, cần nhân bản file đó trên một vài phục vụ và hệ thống được thiết kế sao cho dễ dàng bổ sung phục vụ. Có thể tính đến các giải pháp khác là sử dụng Cache và bản sao dữ liệu.

e. Khả năng thứ lỗi

Khả năng thứ lỗi thể hiện việc hệ thống không bị sụp đổ bởi các sự cố do các lỗi thành phần (cả phần cứng lẫn phần mềm) trong một bộ phận nào đó.

Việc thiết kế khả năng chịu lỗi của các hệ thống máy tính dựa trên hai giải pháp sau đây:

- Dùng khả năng thay thế để đảm bảo việc hoạt động liên tục và hiệu quả.
- Dùng các chương trình đảm bảo cơ chế phục hồi dữ liệu khi xảy ra sự cố.

Để xây dựng một hệ thống có thể khắc phục sự cố theo cách thứ nhất thì có thể chọn giải pháp nối hai máy tính với nhau để thực hiện cùng một chương trình mà một trong hai máy đó chạy ở chế độ Standby (*không tải* hay *chờ*). Giải pháp này khá tốn kém vì phải nhân đôi phần cứng của hệ thống.

Giải pháp khác nhằm giảm bớt phí tổn là dùng nhiều phục vụ khác nhau cung cấp các ứng dụng quan trọng để các phục vụ này có thể thay thế nhau khi sự cố xuất hiện. Khi không có sự cố thì các phục vụ chạy bình thường (nghĩa là vẫn phục vụ các yêu cầu của khách). Khi xuất hiện sự cố trên một phục vụ nào đó, các ứng dụng khách tự chuyển hướng sang các phục vụ còn lại. Với cách thứ hai thì phần mềm phục hồi được thiết kế sao cho trạng thái dữ liệu hiện thời (trạng thái trước khi xảy ra sự cố) có thể được khôi phục khi lỗi được phát hiện. Chú ý rằng với cách thức này, một mặt thì cùng một dịch vụ có thể được sẵn sàng trên nhiều máy và mặt khác, trên một máy lại có sẵn một số dịch vụ khác nhau.

Hệ phân tán cung cấp khả năng sẵn sàng cao để đối phó với các sai hỏng phần cứng. Khả năng sẵn sàng của hệ thống được đo bằng tỷ lệ thời gian mà hệ thống sẵn sàng làm việc so với thời gian có sự cố. Khi một máy trên mạng sai hỏng thì chỉ có công việc liên quan đến các thành phần sai hỏng bị ảnh hưởng. Người sử dụng có thể chuyển đến một trạm khác nếu máy họ đang sử dụng bị hỏng, một QT phục vụ có thể được khởi động lại trên một máy khác.

f. Tính trong suốt

Như đã được trình bày trong chương 1, tính trong suốt là tính chất căn bản của hệ phân tán. Tính trong suốt của hệ phân tán được hiểu như là *sự che khuất đi các thành phần riêng biệt của hệ thống máy tính (phần cứng và phần mềm) đối với người sử dụng* và những người lập trình ứng dụng. Người sử dụng có quyền truy cập đến dữ liệu đặt tại một điểm dữ liệu ở xa một cách tự động nhờ hệ thống mà không cần biết đến sự phân tán của tất cả dữ liệu trên mạng. Hệ thống tạo cho người dùng cảm giác là dữ liệu được coi như đặt tại máy tính cục bộ của mình. Các thể hiện điển hình về tính trong suốt của HĐH phân tán được trình bày trong phần sau.

II.1.2. Mục tiêu thiết kế hệ điều hành phân tán

Các đặc điểm của hệ phân tán cần được tính đến khi thiết kế HĐH phân tán. Mục tiêu thiết kế HĐH phân tán tương đồng với mục tiêu thiết kế HĐH nói chung và cần được xem xét theo hai góc độ: góc độ của người sử dụng và góc độ của nhà cung cấp HĐH. Trong thiết kế HĐH phân tán, những mục tiêu chung nhất theo cả hai góc độ này là cung cấp một mô hình đơn giản hướng tới một hệ thống hiệu quả (efficient), mềm dẻo (linh hoạt - flexible), nhất quán (consistency), mạnh mẽ (robust). Nội dung của bốn mục tiêu thiết kế này cũng bao gói được phần lớn các tính chất của hệ phân tán mà đã được giới thiệu trong mục trước.

Do tính chất "phân tán" vật lý (tài nguyên phân tán, truyền thông mức cao, đa dạng hơn các lỗi thành phần) cho nên HĐH phân tán hoạt động phức tạp hơn, cũng có nghĩa là việc thi hành các mục tiêu trên đây là phức tạp và khó khăn hơn.

- *Tính hiệu quả*

Tính hiệu quả trở nên phức tạp hơn so với HĐH tập trung do phải tính đến chi phí phải trả cho bài toán truyền thông mà trước đây trong HĐH tập trung đã bỏ qua yếu tố này. Truyền thông CTĐ trong môi trường phân tán địa lý dẫn đến độ trễ tới hàng micro giây, mili giây thậm chí là hàng giây và tạo ra một yếu tố phức tạp trong việc đánh giá mức độ hiệu quả của hệ thống.

Nguồn gốc của "độ trễ" là do bổ sung nhiều yếu tố mới vào HĐH phân tán so với HĐH tập trung, đó là độ trễ do nhân bản dữ liệu, độ trễ do tính toán đến tổng phí theo các giao thức truyền thông ở các mức độ khác nhau và sự phân tán tải của hệ thống.

Độ trễ do nhân bản dữ liệu là khá rõ ràng và hiển nhiên. Nhân bản dữ liệu là việc tạo thêm các bản sao dữ liệu từ nơi khác tới vị trí xử lý nhằm mục đích tăng tốc độ truy nhập dữ liệu. Tuy nhiên nhân bản dữ liệu cũng đòi hỏi chi phí phải trả gồm thời gian sao dữ liệu và thời gian đảm bảo yếu tố nhất quán của dữ liệu được nhân bản. Không thể đặt ra giải pháp nhằm hạn chế nhân bản dữ liệu. Tuy nhiên, việc truyền thông mức ngôn ngữ hay HĐH nên làm thật hiệu quả và giao thức truyền thông mức mạng nên làm cho thật tốt. Khi lưu ý đến phân bố tải hệ thống thì những vấn đề như hiện tượng thắt cổ chai hoặc tắc nghẽn hoặc trong mạng vật lý hoặc trong thành phần phần mềm bắt buộc phải được địa chỉ hóa. Các ứng dụng (hệ thống hoặc người dùng) có thể tiến thêm một bước là QT phân tán cần được cấu trúc tốt chẳng hạn như tính toán và truyền thông có thể được cân bằng tải và gối lên nhau một cách hợp lý. Một thuật toán lập lịch tối ưu trong HĐH tập trung có thể không trở thành thuật toán tốt khi áp dụng trong HĐH phân tán. Việc phân tán các QT sao cho hệ thống được cân bằng: các CPU dùng cho xử lý, các đường truyền thông được phát huy cao nhất có thể có.

Hai thông số quan trọng đánh giá hiệu quả hệ phân tán là độ tăng tốc và thông lượng hệ thống. Độ tăng tốc (*speedup*) được hiểu là thời gian hoàn thiện QT là nhanh hay chậm. Thông lượng (*throughput*) được hiểu là số QT đồng thời được xử lý tại một thời điểm. Việc nâng cao hai thông số này thông qua việc lập lịch các QT phân tán, chia xẻ tải và hệ thống truyền thông cần được thiết kế tốt.

•Tính mềm dẻo

Theo cách nhìn của người sử dụng, tính mềm dẻo được thể hiện thông qua tính thân thiện của hệ thống, tính tự do của người dùng khi sử dụng hệ thống. Tính thân thiện được hiểu rất rộng như dễ dàng sử dụng giao diện hệ thống, khả năng ánh xạ quá trình tính toán trong không gian bài toán tới hệ thống. Tiếp cận hướng đối tượng là chiến lược phổ biến để hoàn thành mục tiêu này. Tính thân thiện cũng liên kết với các tính chất nhất quán và tính tin cậy. Các hệ thống nhất quán và đáng tin cậy không có những hạn chế vô lý. Nó cần cung cấp môi trường hoạt động thích hợp trong đó các tool và dịch vụ mới dễ dàng được xây dựng.

Theo cách nhìn của hệ thống, tính mềm dẻo là năng lực của hệ thống để tiến hóa và di trú. Các tính chất mấu chốt là môđun, co giãn, khả chuyển và liên thao tác. Trong những trường hợp khác, các tính chất này có độ quan trọng riêng trong hệ phân tán do hầu hết các hệ thống sử dụng các thành phần phần cứng và phần mềm hỗn tạp. Một mặt, chúng ta mong muốn có một quyền tự trị địa phương, nhưng mặt khác, chúng ta lại muốn cùng cộng tác thành một hệ liên kết chặt chẽ, và chính điều này đã dẫn đến hạn chế nào đó tới chúng ta. Chính từ hai mong muốn có vẻ đối lập nhau này đưa đến giải pháp dung hòa trong việc giải quyết tính mềm dẻo của hệ phân tán.

•Tính nhất quán

Tính nhất quán trở nên khó khăn hơn khi thi hành trong hệ phân tán: thiếu vắng thông tin toàn cục, tiềm tàng nhân bản và phân hoạch dữ liệu mạnh, khả năng xẩy ra lỗi thành phần, mối liên quan phức tạp các môđun thành phần; tất cả các điều đó đều tham gia vào sự thiếu nhất quán của hệ thống. Theo phương diện người dùng, một hệ thống là nhất quán nếu như có được tính đồng nhất khi sử dụng và ứng xử hệ thống có thể khẳng định trước. Hơn nữa, hệ thống phải đủ năng lực duy trì tình trạng toàn vẹn nhờ cơ chế điều khiển đồng thời chính xác và các thủ tục kiểm soát lỗi và khôi phục. Điều khiển nhất quán trong dữ liệu và file (hoặc CSDL trong hệ thống định hướng giao dịch) là những vấn đề còn được bàn luận trong hệ thống file phân tán.

•Tính mạnh mẽ

Bài toán tính mạnh mẽ càng trở nên quan trọng hơn trong hệ thống phân tán: lỗi kết nối truyền thông, lỗi tại nút xử lý và lỗi trong các QT Client/Server là thường xuyên hơn so với hệ thống máy tính tập trung. Quy tắc nào cần được hệ HĐH tuân thủ trong những trường hợp, chẳng hạn như một thông điệp hỏi/đáp bị mất hoặc nút xử lý hoặc phục vụ bị đổ vỡ ? Tính mạnh mẽ về khía cạnh thứ lỗi được hiểu rằng hệ thống đủ năng lực tự khởi động lại tới trạng thái mà tại đó tính toàn vẹn của hệ thống đã được bảo quản mà chỉ với một độ giảm sút hiệu năng một cách hợp lý. Để có tính mạnh mẽ, hệ thống nên được trang bị cơ chế kiểm soát được tình huống khác thường (thậm chí chưa phải là lỗi rõ ràng) và lỗi, chẳng hạn như thay đổi tôpô hệ thống, độ trễ thông điệp lớn, hoặc sự bất lực khi định vị phục vụ. Tính mạnh mẽ cũng nên được mở rộng để phủ được khía cạnh an toàn đối với người dùng và hệ thống. Tính tin cậy, bảo vệ và điều khiển truy nhập là trách nhiệm của HĐH phân tán.

II.2. Tính trong suốt trong hệ phân tán

Tính chất mấu chốt nhất phân biệt hệ phân tán với các hệ thống khác là tính trong suốt, thuật ngữ thường xuyên được nhắc trong các hệ thống phân tán. Nó là mục tiêu thúc đẩy việc che khuất đi những chi tiết phụ thuộc hệ thống mà không thích hợp đối với người dùng trong mọi hoàn cảnh và tạo ra một môi trường thuần nhất cho người dùng. Nguyên lý này đã được thực tế hóa khi thiết kế hệ thống máy tính qua một thời gian

dài. Tính trong suốt trở nên quan trọng hơn trong hệ thống phân tán và thực hiện khó khăn hơn chính từ tính hỗn tạp của hệ thống.

Sự che khuất thông tin phụ thuộc hệ thống khỏi người dùng dựa trên việc cân bằng giữa tính đơn giản và tính hiệu quả. Một cách đáng tiếc, hai tính chất này là xung đột nhau. Bởi vậy, mong muốn một mục tiêu trong suốt hoàn toàn là không thích hợp. Hệ phân tán tốt là cố gắng đạt được tính trong suốt cao nhất có thể được. Tương tự như khái niệm "ảo" trong HĐH và "trừu tượng" trong ngôn ngữ lập trình, mục tiêu của tính trong suốt là cung cấp một cái nhìn lôgic thống nhất của một hệ thống vật lý hỗn tạp nhờ việc rút gọn hiệu quả việc nhận biết hệ thống vật lý tới cực tiểu (nói riêng, theo khía cạnh chia cắt vật lý của các đối tượng và điều khiển trong hệ thống).

• Tính trong suốt thể hiện trong nhiều khía cạnh, dưới đây là một số khía cạnh điển hình nhất:

- Trong suốt truy nhập: Truy nhập đối tượng địa phương/toàn cục theo cùng một cách thức. Sự tách rời vật lý của các đối tượng hệ thống được che khuất tới người dùng.

- Trong suốt định vị (còn được gọi là trong suốt tên): Người dùng không nhận biết được vị trí của đối tượng. Đối tượng được định vị và chỉ dẫn theo tên lôgic trong một hệ thống thống nhất.

- Trong suốt di trú (còn được gọi là độc lập định vị): là tính chất bổ sung vào trong suốt định vị theo nghĩa không những đối tượng được chỉ dẫn bằng tên lôgic mà đối tượng còn được di chuyển tới định vị vật lý khác mà không cần đổi tên.

- Trong suốt đồng thời: cho phép chia xẻ đối tượng dùng chung không gặp tranh chấp. Nó tương tự như khái niệm phân chia thời gian theo nghĩa khái quát.

- Trong suốt nhân bản: đưa ra tính nhất quán của đa thể hiện (hoặc vùng) của file và dữ liệu. Tính chất này quan hệ mật thiết với trong suốt đồng thời song được cụ thể hơn vì file và dữ liệu là loại đối tượng đặc biệt,

- Trong suốt song song: cho phép các hoạt động song song mà người dùng không cần biết hoạt động song song đó xẩy ra như thế nào, ở đâu và khi nào. Tính song song có thể không được người dùng đặc tả.

- Trong suốt lỗi: cung cấp khả năng thứ lỗi của hệ thống được hiểu là lỗi trong hệ thống có thể được biến đổi thành sự giảm hiệu năng hệ thống một cách mềm dẻo hơn chứ không phải chỉ là làm cực tiểu sự đổ vỡ và nguy hiểm đối với người dùng,

- Trong suốt hiệu năng: cố gắng giành được tính nhất quán và khẳng định (không cần thiết ngang bằng) mức độ hiệu năng thậm chí khi thay đổi cấu trúc hệ thống hoặc phân bố tải. Hơn nữa, người dùng không phải chịu sự chậm trễ hoặc thay đổi quá mức khi thao tác từ xa. Trong suốt hiệu năng còn được thể hiện là hiệu năng hệ thống không bị giảm theo thời gian.

- Trong suốt kích thước: liên quan đến tính mềm dẻo và tiềm tàng. Nó cho phép sự tăng trưởng của hệ thống được che khuất đối với người sử dụng. Kích thước hệ thống không tạo ra tác động đối với nhận thức của người dùng.

- Trong suốt duyệt lại chỉ dẫn rằng sự tăng trưởng hệ thống theo chiều dọc là tỷ lệ nghịch với sự tăng trưởng hệ thống theo chiều ngang. Sự duyệt lại phần mềm bị che khuất đối với người dùng. Trong suốt duyệt lại cũng được hiểu như trong suốt phân đoạn.

• Sau đây là một ví dụ giải thích. Trước đây khi điện thoại còn chưa phổ biến (điện thoại liên tỉnh hiếm hoặc rất đắt), dùng điện thoại nếu khoảng cách ngắn còn dùng thư nếu ở xa để liên lạc với người quen. Trường hợp này vi phạm tính trong suốt truy nhập

do phương pháp liên lạc là không đồng nhất. Khi điện thoại liên tỉnh phát triển, giá cả giảm, nếu dùng điện thoại cố định thì liên lạc số điện thoại nội tỉnh khác với liên lạc với số điện thoại tỉnh ngoài (chẳng hạn, bổ sung thêm mã tỉnh). Trường hợp này vi phạm trong suốt định vị. Nếu một người chuyển chỗ ở phải thay số điện thoại mới (không thể dùng số điện thoại cũ) là vi phạm tính trong suốt di trú.

Một phương pháp liên lạc lý tưởng là trên phạm vi toàn hệ thống, có thể "gặp" đối tượng bất kỳ chỉ bởi tên toàn cục (ký hiệu hoặc số) chẳng hạn như số chứng minh nhân dân. Tiếp tục các ví dụ trên đây, nếu mọi người chỉ sử dụng điện thoại di động để liên lạc với nhau thì hệ thống như vậy được coi là thỏa mãn các tính chất trong suốt truy nhập, định vị và di trú.

Từ phân tích trên đây, nhận thấy rằng trong suốt truy nhập, định vị, và di trú có quan hệ gần gũi nhau.

Trong suốt song song, đồng thời, và hiệu năng được thiết kế nhằm che chắn sự quản lý các hoạt động đồng thời đối với các người dùng, dựa trên các quan hệ nội tại người dùng (intrauser), liên người dùng (interuser) và liên nút (internode). Cho phép thực hiện đồng thời ở các mức thực hiện khác nhau: nội tại một người dùng, giữa các người dùng, và giữa các nút phân tán. Trong suốt nhân bản và trong suốt lỗi có quan hệ với việc duy trì tính toàn vẹn hệ thống. Trong suốt kích thước và trong suốt duyệt lại cung cấp sự biến đổi uyển chuyển của hệ thống theo sự tăng trưởng về phần cứng và phần mềm.

Danh sách trong suốt được mô tả trên đây không phải là toàn diện. Tuy nhiên, danh sách này thích hợp với hệ phân tán. Chúng cũng được phân lớp khi xem xét mối quan hệ với các mục tiêu thiết kế hệ điều hành. Trong suốt đồng thời và hiệu quả cung cấp tính hiệu quả. Trong suốt truy nhập, định vị, di trú và kích thước liên quan đến tính mềm dẻo. Tính nhất quán liên quan tới trong suốt truy nhập, nhân bản và trong suốt hiệu năng. Cuối cùng, các trong suốt lỗi, nhân bản, và kích thước liên quan tới tính mạnh mẽ của hệ thống. Bảng 2.1 cho mối liên hệ giữa mục tiêu của hệ thống với tính trong suốt.

Bảng 2.1 . Phân lớp các tính trong suốt theo mục tiêu hệ thống

Mục tiêu của hệ thống	***Tính trong suốt***
Hiệu quả	đồng thời / song song / hiệu năng
Mềm dẻo	truy nhập / định vị / di trú / kích thước / duyệt lại
Bền vững	truy nhập / nhân bản / hiệu năng
Mạnh mẽ	thứ lỗi / nhân bản / kích thước / duyệt lại

Thi hành HĐH phân tán và các thuật toán điều khiển phân tán tương ứng liên quan chạt chẽ tới việc thực hiện các tính trong suốt này. Nói tóm lại, hệ phân tán cung cấp sự tách rời vật lý của các đối tượng, tính trong suốt được dùng để che khuất đi tác động của sự chia tách vật lý này. Kết quả cuối cùng là người dùng nhìn hệ đa máy tính như một hệ máy tính đơn lôgic.

Bàn luận trên đây về tính trong suốt dựa trên các tính chất của hệ thống đáng mong muốn theo quan điểm của cả người dùng lẫn hệ thống. Các bài toán chính trong HĐH phân tán cũng được phân lớp theo tính trong suốt và được trình bày trong bảng 2.2. Nói tóm lại, mục tiêu của HĐH phân tán là cung cấp môi trường tính toán hiệu năng cao và mạnh mẽ với việc nhận biết ít nhất về quản lý và điều khiển của các tài nguyên hệ thống phân tán.

Bảng 2.2. Vấn đề của hệ phân tán và tính trong suốt

Vấn đề chính của hệ thống	***Tính trong suốt***
Truyền thông Đồng bộ Thuật toán phân tán	Trong suốt liên thao tác và điều khiển
Lập lịch quá trình Nắm giữ bế tắc Cân bằng tải	Trong suốt hiệu năng
Lập lịch tài nguyên Chia xẻ file Điều khiển đồng thời	Trong suốt tài nguyên
Kiểm soát lỗi Cấu hình Thu gọn	Trong suốt lỗi

II.3. Các dịch vụ

HĐH được coi là "nhà" cung cấp dịch vụ và để thiết kế hiệu quả các dịch vụ này thì chúng nên được tổ chức và xây dựng theo phân cấp. Như vậy, dịch vụ được tạo nên từ các dịch vụ đã có và chúng ta nhận được một cấu trúc nhiều mức dịch vụ: dịch vụ nguyên thủy, dịch vụ từ phục vụ hệ thống và dịch vụ gia tăng giá trị.

2.3.1 Dich vu nguyên thủy

Dịch vụ nguyên thủy là mức thấp nhất trong hệ thống các mức dịch vụ, chúng là những dịch vụ cơ bản nhất, chúng tồn tại trong nhân của HĐH mỗi nút trong hệ thống.

Dịch vụ nguyên thủy, bắt buộc phải được đưa vào nhân của HĐH: điều này tương ứng với cách tiệm cận "nhân tối thiểu" (vi nhân) của HĐH tập trung. Ba dịch vụ (chức năng) cơ bản mà nhân buộc phải cung cấp được định danh như sau:

- *Dịch vụ truyền thông*: Trong hệ phân tán, truyền thông giữa các QT được thực hiện nhờ CTĐ, một tập các *dịch vụ nguyên thủy* gửi và nhận buộc phải được xác định và thi hành. Các dịch vụ nguyên thủy này truyền tin theo kênh lôgic.

- Gửi và nhận có thể đồng bộ hoặc dị bộ. *Truyền thông đồng bộ* thêm vào phục vụ mục đích truyền thông, được phát triển từ đồng bộ truyền thông liên QT (tại một nút) nhằm giúp ích cho truyền thông liên nút. Nếu CTĐ chỉ theo nghĩa tương tác QT, đồng bộ QT phải dựa vào truyền thông hoặc chính ngữ nghĩa đồng bộ của truyền thông hoặc bởi các phục vụ đồng bộ nào đó dựa trên CTĐ. Trước hết cần có các dịch vụ nguyên thủy đồng bộ (syschronous primitive), còn được gọi là dịch vụ kết khối (blocking primitive). Đối ngẫu với chúng là nguyên thủy dị bộ (asyschronous primitive) hay dịch vụ không kết khối (nonblocking primitive). Ngoài ra còn có các cặp dịch vụ nguyên thủy buffer (buffered primitive/ unbuffered primitive) và cặp các dịch vụ nguyên thủy tin cậy (reliable primitive/ unreliable primitive).

- Vì yêu cầu che đậy sự phụ thuộc vật lý trong hệ phân tán, mô tả bộ xử lý đa thành phần như là một phục vụ QT là phù hợp hơn. *Dịch vụ QT* quản lý việc phát sinh, loại bỏ và điều chỉnh các QT bằng cách định vị các tài nguyên cần thiết, chẳng hạn như bộ nhớ và thời gian xử lý. Việc giải đáp vấn đề bộ xử lý là cục bộ hay từ xa, yêu cầu bao nhiêu bộ xử lý rồi tới các QT là trong suốt.

Phục vụ QT tương tác với các phục vụ QT khác thông qua truyền thông từ xa và đồng bộ.

2.3.2. Dịch vụ từ phục vụ hệ thống

Có rất nhiều dịch vụ tuy rất cần thiết song không bắt buộc phải đưa vào nhân, các dịch vụ này được các phục vụ hệ thống cung cấp. Dưới đây là một số dịch vụ điển hình nhất được liệt kê theo mức độ quan hệ với hệ thống phân tán.

Chức năng che giấu đối tượng vật lý bằng tên lôgic đòi hỏi tồn tại cơ chế ánh xạ tên lôgic thành đối tượng vật lý. Địa chỉ của một QT hay định vị một file có thể thu được theo cách ad-hoc (không dự tính trước), nhưng nói chung là qua xem xét của phục vụ tên hoặc phục vụ thư mục. *Phục vụ tên* thường được dùng để định vị (định danh) người dùng, QT, hoặc máy còn *phục vụ thư mục* thường được dùng để liên kết với file hoặc cổng truyền thông.

Nếu không thể định vị được phục vụ thì phục vụ là vô dụng vì vậy mọi phục vụ cần được định vị bởi dịch vụ tên. Như vậy phục vụ tên là phục vụ *thiết yếu nhất* trong hệ phân tán. Việc thu được địa chỉ và định vị từ phục vụ tên phụ thuộc hệ thống và buộc phải được giải thích thành đường truyền thông trước khi đối tượng được truy nhập. Các dịch vụ giải thích này, gồm chọn đường đi và chọn lộ trình thực sự của thông tin, là các dịch vụ được cung cấp bởi *phục vụ mạng*. Phục vụ mạng được trong suốt theo mức độ HĐH. Việc truyền phát TĐ trong mạng mà chưa dùng đến khả năng hỗ trợ hiệu quả của phần cứng cần tới một *phục vụ truyền phát* hay *phục vụ khuyếch tán* trong HĐH.

Phục vụ quan trọng tiếp theo là *phục vụ thời gian*. Đồng hồ (tổng quát hơn là *bộ thời gian*) được dùng để đồng bộ và lập lịch các hoạt động phần cứng và phần mềm trong mọi hệ thống máy tính. Về mặt lý thuyết thì không thể đưa vào hay chấp nhận một thông tin đồng hồ tổng thể tuyệt đối. Thậm chí có tồn tại một đồng hồ trung tâm (kiểu thời gian Greenwich) thì độ lệch thời gian vẫn xuất hiện do độ trễ khi tiếp nhận và ghi thông tin thời gian. Trong hệ phân tán, độ lệch này càng lớn do độ lệch truyền thông giữa các QT dài hơn. Tại nức HĐH, có hai kiểu sử dụng thông tin thời gian điển hình để đồng bộ các QT: (1) đòi hỏi một xấp xỉ gần gũi thời gian (chẳng hạn thời đoạn CPU phục vụ một QT) với đồng hồ thời gian thực và (2) sử dụng đồng hồ thời gian nhân tạo nhằm duy trì quan hệ nhân quả thứ tự sự kiện (sự kiện nào xuất hiện trước trong hai sự kiện). Tương ứng, chúng được gọi là đồng hồ vật lý và đồng hồ lôgic. Mục đích dùng đồng hồ vật lý là đảm bảo tính đồng bộ thao tác hoặc đòi hỏi rằng thao tác thực sự được giải quyết tại một thời điểm thời gian thực. Thời gian buộc phải phù hợp nhờ một độ đo thực sự nào đó song việc có gần gũi với thời gian thế giới thực hay không là không quan trọng. Đồng bộ các QT sử dụng đồng hồ lôgic do cần duy trì một thứ tự tổng thể việc xuất hiện các sự kiện nhằm tin chắc chắn vào tính đúng đắn về sự phụ thuộc lẫn nhau của các thao tác. Phục vụ thời gian cho đồng hồ vật lý dựa trên việc xấp xỉ tốt nhất đồng hồ thời gian "thực". Giải pháp thực hiện đồng bộ đồng hồ logic là hợp lý và được thực hiện bằng đồng hồ lôgic Lamport *xuất hiện trước*.

Phục vụ tên và phục vụ thời gian là các *phục vụ thông tin*. Thông thường, còn đòi hỏi thêm các phục vụ hệ thống khác nhằm *quản lý tài nguyên hệ thống được chia xẻ*. Ví dụ quen thuộc đó là *phục vụ file* và *phục vụ in*. Phục vụ file có thể được nhân bản hay tách nếu file không biến đổi. Các phục vụ có thể được cấu trúc thứ bậc, chẳng hạn, phục vụ file có thể chứa các phục vụ con thư mục hoặc an toàn để điều khiển truy nhập và xác nhận quyền; phục vụ QT có thể bổ sung thành một phục vụ di trú với sự cộng tác của phục vụ QT để thuận tiện chuyển QT từ nút này sang nút khác. Trong một hệ thống phân tán rộng lớn mà các QT truyền thông có thể nghi ngại nhau, cần đến một phục vụ xác định tin cậy để xác định định danh quá trình. Phục vụ hệ thống cung cấp những dịch vụ cơ sở để quản lý quá trình, file, truyền thông quá trình.

2.3.3. Dịch vụ gia tăng giá trị

Dịch vụ gia tăng giá trị là dịch vụ không nhất thiết được thi hành trong hệ thống phân tán nhưng lại rất hữu ích để hỗ trợ các ứng dụng phân tán. Về phía người dùng, sử dụng các ứng dụng phân tán được thúc đẩy không chỉ với mong muốn làm tăng hiệu năng tính toán và tăng việc thứ lỗi mà còn cần được tạo ra các hoạt động cộng tác. Mặt khác, cần quan tâm tới khái niệm nhóm các quá trình tương tác. Nhóm tính toán tựa như một tổ chức xã hội và cần thiết phải được quản lý.

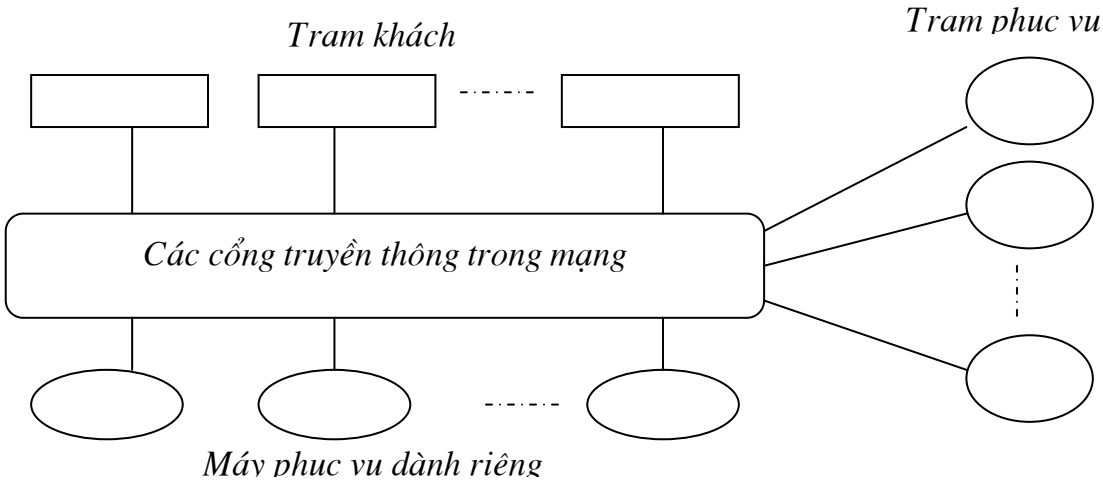
Phục vụ nhóm quản lý việc khởi tạo và kết thúc các hoạt động nhóm, bao gồm cả địa chỉ nhóm và truyền thông. Nó thực hiện cả chức năng quản lý thông tin như tính thành viên, chính sách kết nạp và đặc quyền của thành viên. Quyết định của nhóm về việc thu nhận hoặc rời khỏi nhóm được thi hành bởi người lãnh đạo nhóm hoặc sự nhất trí của toàn thể thành viên trong nhóm. Nghi thức nhóm đảm bảo tính chất xác định của nhóm. Cấu trúc nhóm biến động theo các ứng dụng. Ví dụ, nhóm tin trên mạng, nhóm hội thảo phân tán, nhóm soạn thảo đồng thời, nhóm tính toán phân tán trải từ kết nối lỏng đến kết nối rất chặt (tăng dần theo mức độ gắn kết) với các quy tắc chi phối cứng rắn. Khi một ứng dụng nhóm trở thành chuẩn tắc thì nó được trộn vào hệ thống trở thành một dịch vụ mới. Theo nghĩa này, phục vụ hội thảo phân tán, phục vụ soạn thảo đồng thời là cần đưa vào hệ thống. Phục vụ Web là một ví dụ phục vụ giá trị giá tăng chuẩn tắc.

II.4. Mô hình kiến trúc hệ điều hành phân tán

Các dịch vụ phân tán được mô tả thông qua các đặc tả chức năng. Thi hành các dịch vụ phân tán phụ thuộc vào *kiến trúc hệ thống* và *mạng truyền thông hạ tầng*. Mức trên, kiến trúc hệ thống được mô tả một cách trừu tượng thông qua các thành phần chính và mối quan hệ của các thành phần này. Mức dưới, kiến trúc mạng đặc tả các phương tiện truyền thông. Tồn tại rất nhiều mô hình hệ thống và kiến trúc mạng, mà dưới đây là các mô hình được dùng phổ dụng nhất và được minh họa thông qua các khái niệm phần mềm.

2.4.1. Kiến trúc hệ thống phân tán (mức trên)

Theo nghĩa trừu tượng nhất, thế giới HĐH tập trung chỉ bao gồm hai kiểu thực thể chính là QT và file. Theo giả thiết của một mạng, HĐH phân tán bắt buộc bổ sung thành phần thứ ba, đó là đường truyền thông (hay cổng) trên đó cho phép tính trong suốt của hệ thống.

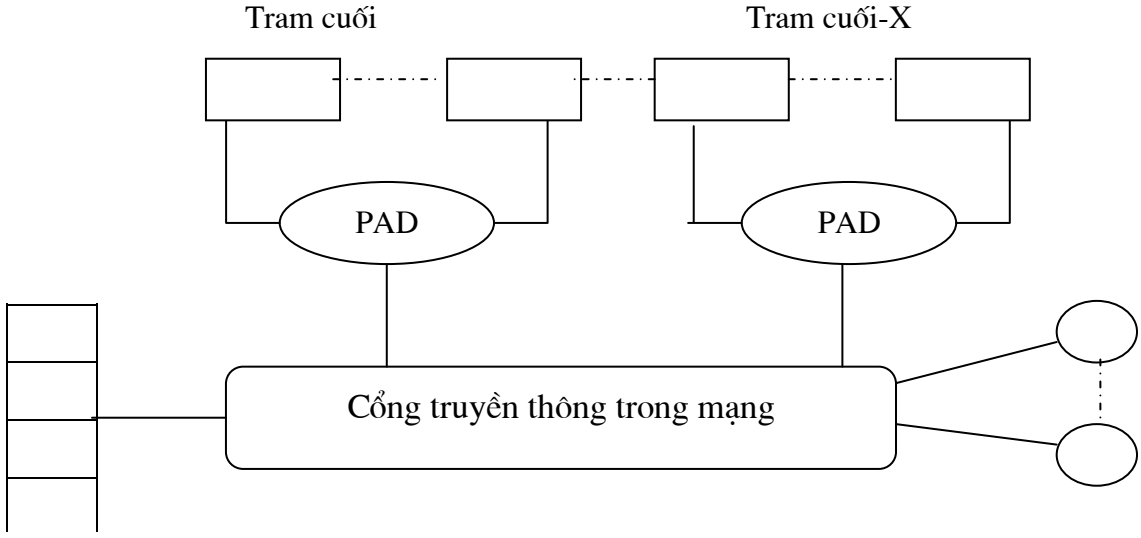


Hình 2.1. Mô hình trạm - phục vụ

Như vậy, trong HĐH phân tán, ba thực thể chính là QT, File và đường truyền thông. Phần cứng nhìn thấy được là các trạm làm việc và mạng. Hơn nữa, người dùng không quan tâm đến các thành phần vật lý, ngoại trừ giá cả, hình dáng và hiệu năng của chúng. QT người dùng hoặc đòi hỏi hoặc cung cấp dịch vụ. Một số trạm làm việc được dành riêng để cung cấp các dịch vụ (theo lý do địa lý hoặc hiệu năng, gọi là máy trạm phục vụ dành riêng). Đây là mô hình trạm làm việc-phục vụ (workstation - server) được trình bày trong hình 2.1.

Trong mô hình trạm - phục vụ, máy trạm có thể đóng vai trò như một máy tính độc lập hoặc như một phần của mạng toàn bộ. Nó cung cấp năng lực một bộ xử lý cục bộ và một giao diện mạng. Một số trạm làm việc không có đĩa, mọi dịch vụ file và khởi động được hỗ trợ bởi hệ thống file mạng. Sự tồn tại đĩa là trong suốt, ngoại trừ tiếng lạch cạch định kỳ của ổ đĩa, cho biết máy tính đang tồn tại. Nếu QT được thực hiện từ xa thì điều phân biệt giữa một trạm làm việc với một trạm cuối trở nên mập mờ.

Trong mô hình trạm - phục vụ, đa số các trạm làm việc nhàn rỗi. Điều này đưa đến một yêu cầu là nên chăng tập trung mọi năng lực xử lý vào một vị trí và cho phép người dùng chỉ dùng trạm cuối (hỗ trợ tốt phần cứng và phần mềm hiển thị). Năng lực xử lý có thể được tận dụng tốt hơn trên một nền tảng yêu cầu. Quan niệm này dẫn tới mô hình xâu bộ xử lý (processors - pool), được chỉ ra trong hình 2.2. Đặc trưng chính của mô hình này là chia xẻ tính toán, bổ sung vào chia xẻ tài nguyên.



Hình 2.2. Mô hình xâu - bộ xử lý

Trong mô hình xâu-bộ xử lý, người dùng được phép truy nhập vào hệ thống máy tính đơn ảo nhờ vào các trạm cuối thông minh, như X-terminal. Tiêu chuẩn "thông minh" của trạm cuối ít nhất phải bao gồm (1) boot máy từ xa, (2) gắn kết (mount) hệ thống file từ xa, (3) chủ động quản lý trạm cuối ảo, (4) năng lực thiết kế và giải thiết kế gói (packet assembling and deassembling - PAD) đối với truyền thông chuyển mạch gói. PAD có thể được thiết kế ngay tại trạm cuối hoặc đứng riêng để có thể có được tính đa thành phần gồm một số lượng lớn các trạm cuối. Định vị file và bộ xử lý do hệ thống thực hiện. Mô hình này có được tính trong suốt song song.

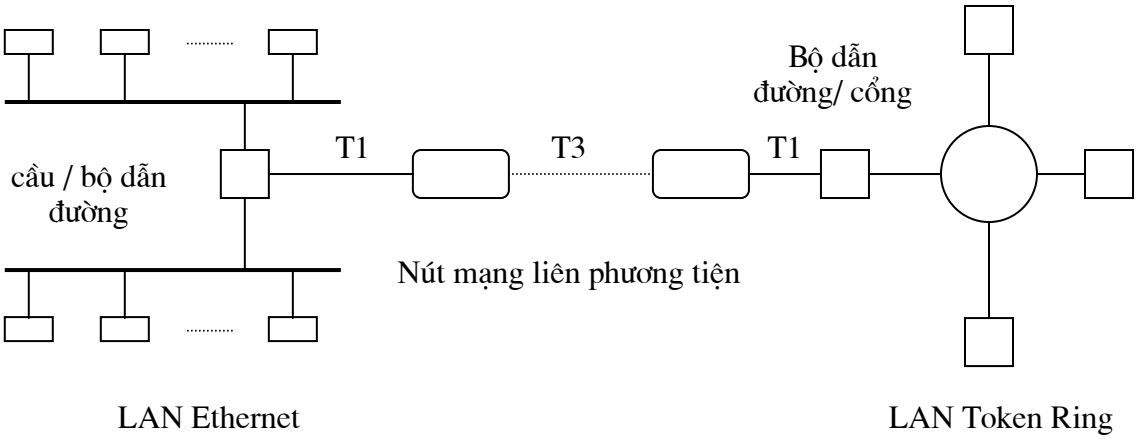
Tồn tại mô hình kiến trúc "lai" là kết quả tích hợp nội dung hai mô hình trên đây. Hiện vẫn còn một số câu hỏi về cách thức tích hợp hai mô hình: Có thể hay không các bộ xử lý trong mô hình trạm - phục vụ được dùng như xâu - bộ xử lý ? Nếu di trú QT được thi

hành hiệu quả thì tại sao lại không làm như vậy. Các QT có thể di trú tới các bộ xử lý nhàn rỗi hoặc tải ít dựa theo chiến lược tải nào đó. Tất nhiên là giải pháp như vậy còn liên quan đến nhiều yếu tố khác nữa.

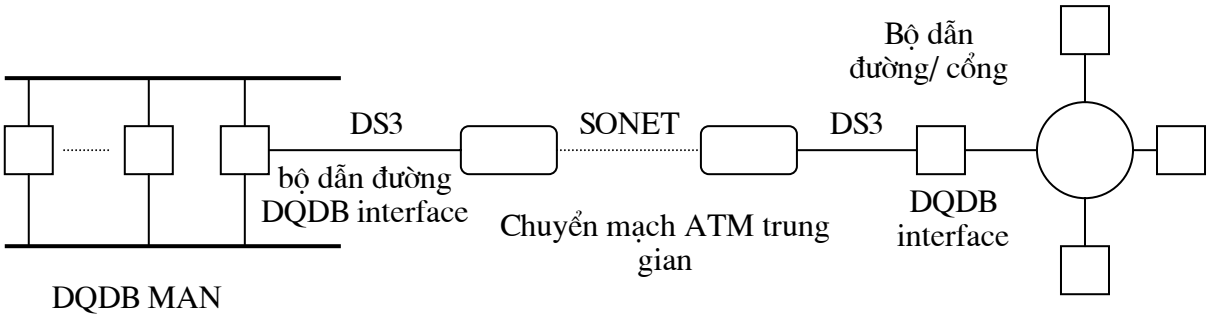
2.4.2. Kiến trúc mạng truyền thông

Tính đa thành phần của hệ thống máy tính trong hệ phân tán dẫn đến đòi hỏi cần kết nối các thành phần này. Nói khác đi, nhu cầu chia xẻ tài nguyên (phần cứng, phần mềm, thông tin ...) từ một tập hợp nhiều máy tính đòi hỏi liên kết các thành phần trong tập hợp đó.

Điều này đã được thực hiện trong mạng với HĐH mạng. Trong hệ phân tán, hạ tầng kết nối tương tự như HĐH mạng. Các thành phần trong hệ thống có thể kết nối điểm-điểm (có kết nối trực tiếp giữa các cặp nút máy tính trong hệ thống) hoặc kênh truyền thông đa điểm (tuyến chia xẻ - bus, hay mạng liên kết nối - switch).



Hình 2.3.a. WAN tốc độ thấp



Hình 2.3.b. WAN hoặc MAN tốc độ cao

Truy nhập các phương tiện truyền thông trong *tuyến chia xẻ* là phân chia thời gian, còn *switch* cung cấp tính đa hợp không gian-thời gian (phân chia cả thời gian lẫn không gian) với chi phí phần cứng và độ phức tạp cao hơn.

Mạng truyền thông dựa trên-bus được dùng rộng rãi trong mạng LAN vì công nghệ đơn giản và tính hoàn thiện của nó. Chuẩn IEEE 802 LAN định nghĩa một số phương tiện dựa trên-bus, bao gồm Ethernet, Token Bus, Token Ring, Fiber Distributed Data Interface (FDDI) và Distributed Queue Dual Buses (DQDB).

Hệ thống switch phổ thông hơn, thường được dùng để thi hành các hệ thống đa xử lý hiệu năng cao và liên kết nối các mạng LAN dọc theo các vật mang dịch vụ truyền thông công cộng. Switch nguyên thủy điển hình cho hệ đa xử lý là mạng liên kết nối ngang hàng hoặc đa mức. Dịch vụ switch công cộng bao gồm các dạng mạng

Intergrated Services Digital Network (ISDN) và ô rơle Switched Multimegabit Data Service (SMDS), và nổi bật là Asynchronous Transfer Mode (ATM). Điều quan trọng dẫn đường dữ liệu không phải do host mà do các switch thi hành. Switch còn đảm nhận một số chức năng mạng.

Các LAN được kết nối thành mạng thành phố (MAN: Metropolitan Area Network) hoặc mạng diện rộng (WAN: Wide Area Network). Mạng được đặc trưng theo yêu cầu năng lực và khoảng cách địa lý của chúng. LAN có năng lực đến 20 Mbps và khoảng cách dưới vài km, được dùng cho một cơ quan cục bộ đơn lẻ. WAN có thể trải trên khoảng cách vài trăm đến vài nghìn km và qua một số nút truyền thông. WAN truyền thống tốc độ thấp, tốc độ dữ liệu từ vài Kbps (kilobits-per-second) và đòi hỏi bộ đệm lưu giữ-chuyển tiếp (store-and-forward). Năng lực điển hình là các kênh T1 56 Kbps và 1.544 Mbps. Đã mở rộng năng lực bằng công nghệ sợi quang đạt từ mức DS1 (1.5 Mbps) tới mức DS3 (45 Mbps) và giao thức mạng quang đồng bộ (Synchrronous Optical NETwork - SONET, 155 Mbps). Tốc độ cao cho phép phát triển WAN và MAN. Dữ liệu trộn bao gồm văn bản, tiếng nói, video được sinh ra từ các LAN. Áp dụng công nghệ chuyển mạch gói nhằm nâng cao tốc độ và kéo dài khoảng cách truyền.

Năng lực và khoảng cách đóng vai trò quan trọng trong việc xác định kiểu ứng dụng có thể thực hiện trên mạng. Để phân biệt, sử dụng tham số ***a*** như tỷ số của độ trễ truyền trên độ trễ chuyển. LAN có tham số ***a*** nhỏ thua nhiều so với MAN hoặc WAN tốc độ cao. Mạng với ***a*** bé là mạng kín, với ***a*** lớn là mạng định hướng truyền thông.

II.5. Các giao thức mạng truyền thông

Sau khi cung cấp một kiến trúc mạng, cần đưa ra các quy tắc và chuẩn để quản lý việc truyền thông. *Giao thức truyền thông* là tập các quy tắc quy định việc trao đổi TĐ nhằm cung cấp dòng thông tin đáng tin cậy và đúng trật tự trong quá trình truyền thông. Để truyền thông tới điểm xa trong đời thường, con người dùng điện thoại và thư. Tương tự, trong hệ thống truyền thông máy tính cũng có hai chiến lược dịch vụ truyền thông là định hướng kết nối (như điện thoại) và không kết nối (như thư).

Giao thức định hướng kết nối đòi hỏi khởi động rõ ràng một kết nối trước khi bắt đầu truyền thông thực sự. Thông điệp được phân phát tin cậy và theo dòng tuần tự. Thao tác giao thức không kết nối làm việc tương tự chuyển thư của ngành bưu chính: không cần thiết thiết lập cuộc kết nối. TĐ được phân phát dựa trên sự cố gắng nhất về thời gian và đường đi và có thể xuất hiện theo thứ tự tùy ý. Như vậy, việc chuyển phát TĐ không kết nối là không bắt buộc điểm gửi và điểm nhận duy trì một kết nối trong suốt quá trình phân phát TĐ và vì vậy, thời gian thực hiện gửi và nhận của mỗi đối tượng tương ứng là nhanh chóng.

Chọn lựa giao thức nào là tuỳ thuộc vào ứng dụng. Nếu truyền File thì hợp lý hơn dùng định hướng kết nối, còn nếu quảng bá trạng thái hệ thống cục bộ thì dùng không kết nối.

Tại mức mạng, hai dịch vụ kiểu này thường được chỉ dẫn như là *vòng ảo* và *gói dữ liệu ảo*. Tại mức truyền thông phần cứng, gọi chúng là *chuyển mạch vòng* (không kết nối) và *chuyển mạch gói* (hướng kết nối). Kết nối trong chuyển mạch vòng là kết nối vật lý cố định. Vòng ảo là kết nối lôgic. Một đường vật lý đơn có thể chuyển một số vòng ảo đa hợp. Có thể mở rộng tính trừu tượng này tới truyền thống hướng kết nối. Truyền thông hướng kết nối được thi hành nhờ một số dịch vụ gói. Dữ liệu thực sự được mang chuyển trong mạng chuyển mạch gói suốt thời gian hệ thống cung cấp việc phân phát các TĐ một cách tin cậy - đúng thứ tự và cho người dùng cảm giác về một kết nối.

Trong hệ thống, truyền thông xẩy ra tại nhiều mức khác nhau. Người dùng CTĐ lẫn nhau. Các máy truyền thông lẫn nhau. Các mạng cũng đòi hỏi cộng tác thông qua truyền thông. Đây là truyền thông điểm-điểm: truyền thông giữa hai thực thể đồng mức. Thiết kế phần mềm và phần cứng truyền thông trong hệ thống máy tính là một nhiệm vụ rộng lớn hơn. Chúng được cấu trúc thành các tầng.

Việc phân tầng truyền thông cần đảm bảo một số nguyên tắc sau:

(1) Một tầng truyền thông tương ứng với một mức trừu tượng,

(2) Mỗi tầng cần thực hiện chức năng hoàn toàn xác định. Việc xác định chức năng của mỗi tầng truyền thông cần phù hợp quy tắc chuẩn hoá quốc tế,

(3) Thông tin đi qua mỗi tầng là ít nhất,

(4) Số tầng phải đủ lớn để các chức năng tách biệt không nằm trong cùng một tầng và đủ nhỏ để mô hình không quá phức tạp,

(5) Một tầng có thể được phân thành các tầng con nếu cần thiết và các tầng con có thể bị loại bỏ,

(6) Hai hệ thống khác nhau có thể truyền thông với nhau nếu chúng bảo đảm những nguyên tắc chung (cài đặt cùng một giao thức truyền thông),

(7) Các chức năng được tổ chức thành một tập các tầng đồng mức cung cấp chức năng như nhau. Các tầng đồng mức phải sử dụng một giao thức chung.

Như vậy, mỗi tầng cần được mô tả chính xác về chức năng (hay dịch vụ). Hai tầng kề nhau có giao diện và giao thức truyền thông rõ ràng giữa chúng. Tầng thấp hơn cung cấp các dịch vụ cho tầng ngay trên nó. Giao thức giữa các tầng là đơn giản hơn và có cấu trúc thuần nhất hơn so với giao thức điểm-điểm vì thực hiện việc trao đổi giữa hai tầng. Cho phép mở rộng đặc điểm chức năng là phân tích được và kiến trúc giao diện đơn giản xác định hoàn toàn để khởi tạo một đặc tả mạng tường minh cho nhiều nhà cung cấp thi hành. Đặc tả mạng được chuẩn hóa được gọi là kiến trúc hệ thống mạng. Các mức trong mạng chuẩn hóa cùng các giao thức tương ứng được gọi là bộ giao thức mạng.

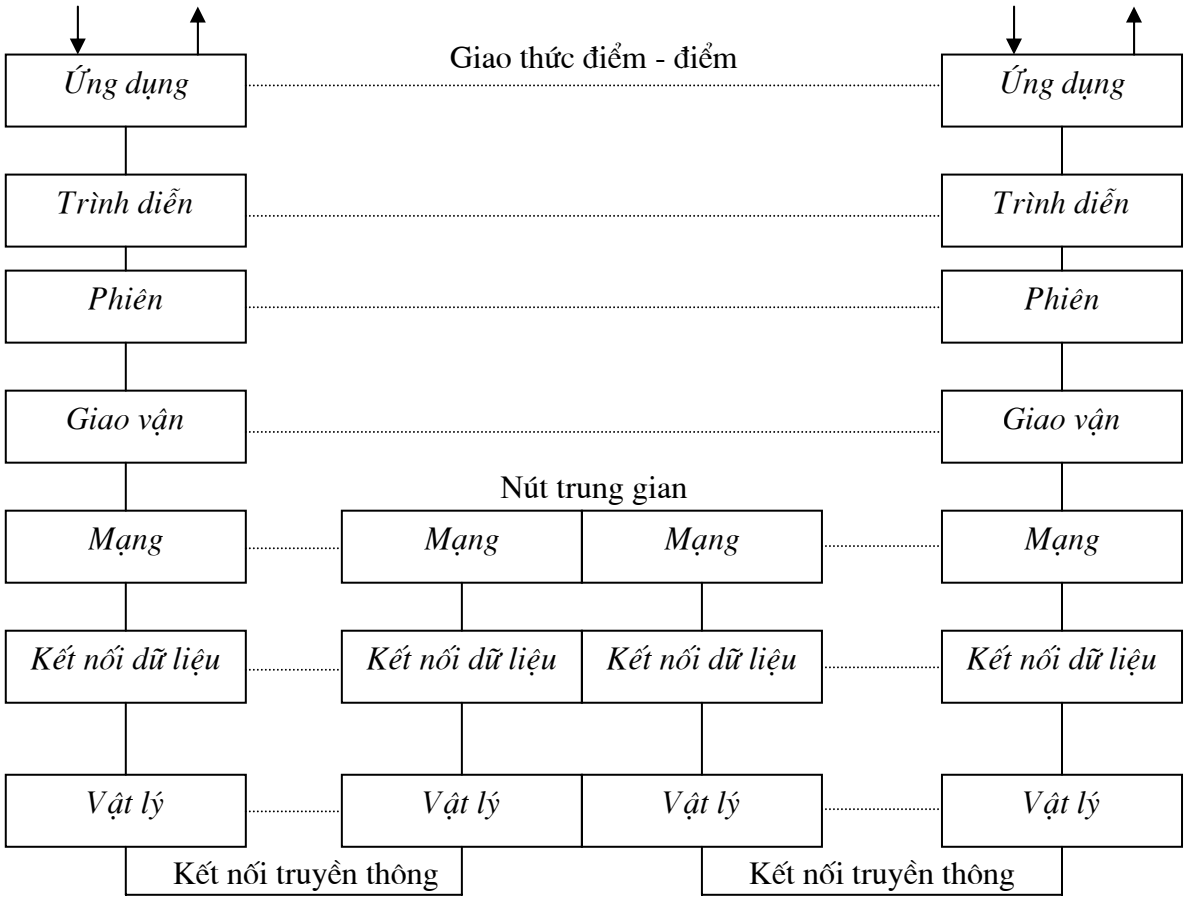
Dưới đây là hai bộ giao thức mạng điển hình nhất: Open Systems Interconnection (OSI) của Tổ chức chuẩn hóa quốc tế (ISO) và Transmission Control Protocol/Internet Protocol (TCP/IP) của Bộ quốc phòng Mỹ.

2.5.1. Bô giao thức OSI

OSI là bộ giao thức bẩy tầng. Một QT truyền thông tới một QT khác ở xa bằng cách chuyển dữ liệu qua bảy tầng, rồi đến tầng vật lý, và cuối cùng đi qua các tầng ở xa theo thứ tự ngược lại. Chi tiết của CTĐ là bị che từ QT truyền thông và các QT này chỉ quan sát được giao thức điểm-điểm. Trong suốt dữ liệu giữa các tầng là đạt được (Hình 2.4).

Tại nút gửi, mỗi tầng nhận một đơn vị dữ liệu giao thức (PDU: Protocol Data Unit) từ tầng ngay trên và gói PDU này với thông tin điều khiển header cho tầng ngang hàng với nó ở điểm nhận. TĐ kết hợp này (TĐ và header) được coi là PDU cho tầng dưới. Trong nhiều trường hợp, PDU cần được phân đoạn do giới hạn kích thước PDU của tầng dưới. Việc phân đoạn và ghép nối lại cần trong suốt ở tầng trên.

Tại nút nhận, thông tin header được tháo ra tương ứng theo từng tầng. Cổng vào ra hoặc nút trung gian chỉ thực hiện việc lưu tạm thời và chuyển tiếp tại ba tầng phụ thuộc mạng thấp nhất. Dưới đây trình bày nội dung các tầng theo thứ tự "từ thấp lên cao".



Hình 2.4. Bộ giao thức bảy tầng OSI

• ***Tầng vật lý*** đặc tả đặc trưng điện tử và cơ học của đường truyền thông vật lý giữa cặp hai nút. Chức năng chính là cung cấp một ống dẫn bit (bit pipe) lôgic tin cậy đối với kênh truyền thông. Tín hiệu điện tử hoặc quang cần được biến đổi thành bit và ngược lại. Ánh xạ tín hiệu thành bit cần đồng bộ bit: phát hiện bắt đầu của bit hoặc dãy bit. Dãy bit hoặc được đồng bộ bit hoặc dị bộ ký tự. Dãy đồng bộ bit là một khối lớn các bit được truyền theo tốc độ thông thường. Phương pháp này cho tốc độ truyền dữ liệu cao hơn và tận dụng đường truyền thông tốt hơn. Dữ liệu dị bộ kí tự là một dãy bít có kích thước nhỏ cố định được truyền dị bộ theo ống dẫn bit. Trạm cuối hướng kí tự tốc độ thấp thường dùng phương pháp này để chuyển dữ liệu. Việc phát và nhận bit lôgic đòi hỏi chuẩn hóa các thuộc tính điện tử và cơ học chẳng hạn như phương pháp mã, kỹ thuật mã hóa gồm ghi và đặc tả vật kết nối. Hai chuẩn kết nối điểm-điểm điển hình là RS232C và X.21. Hơn nữa, modem (MOdunlator/DEModunlator) có thể dùng giữa một cặp cổng RS232C và như vậy trở thành cổng của tuyến vật lý. Modem (đồng bộ hoặc dị bộ) bắt buộc phải chuẩn hóa. Cũng vậy, chia xẻ tuyến dùng chung, chẳng hạn dây đồng trục Ethernet, cũng được chuẩn hóa làm tầng kết nối dữ liệu.

• ***Tầng điều khiển kết nối dữ liệu (DLC)*** đảm bảo truyền tin cậy nhóm bit (được gọi là khuôn - frame). Giao thức kết nối dữ liệu quản lý khởi tạo cấu hình, điều khiển lỗi, tính kế tiếp, điều khiển dòng các khuôn. Cấu hình trình diễn cách thiết lập và kết thúc kết nối và xác định kết nối hai chiều hay một chiều, đồng bộ hay dị bộ. Lỗi gồm có lỗi đường truyền hoặc thiếu/lặp khuôn. Lỗi được phát hiện nhờ cơ chế tổng kiểm tra (checksum) hoặc cơ chế quá hạn (time-out) và được điều chỉnh lại nhờ cơ chế truyền lại hoặc truyền bản đúng của các bản lỗi. Điều khiển dãy dùng số hiệu dãy để duy trì

việc truyền có thứ tự các khuôn. Số hiệu còn được dùng để phát hiện việc sót/lặp khuôn và giúp cho việc điều chỉnh dòng truyền khuôn. Dòng khuôn dữ liệu cần phải được điều chỉnh nếu như nơi nhận không thể theo kịp nơi gửi, có thể do dung lượng quá hạn chế của bộ đệm. Việc phát một khuôn được phép chỉ khi nó rơi vào cửa sổ bộ đệm của người gửi và người nhận. Tất cả chức năng điều khiển này làm tăng chi phí tính theo bit được bổ sung vào khuôn dữ liệu tạo thành đầu và đuôi của khuôn. Kết quả là mỗi giao thức DLC có dạng khuôn xác định để thông dịch đúng đắn các bit trong các trường điều khiển.

Để cấu hình đa điểm, chẳng hạn như tuyến chung, tầng điều khiển kết nối dữ liệu được thi hành với tầng con điều khiển truy nhập trung gian MAC (Medium Access Control) đặt giữa tầng con DLC và tầng vật lý. MAC giải quyết bài toán truy nhập kênh đa tầng.

• ***Tầng mạng***: Do hai tầng thấp nhất (tầng vật lý và tầng kết nối dữ liệu) chỉ liên quan đến một tuyến kết nối. Tầng mạng giải quyết vấn đề gửi các gói (packet) dọc theo mạng thông qua một số đoạn tuyến kết nối. Gói là đơn vị cơ sở truyền dữ liệu trong tầng mạng. Kích thước của gói khác với khuôn của tầng kết nối dữ liệu (kích thước khuôn phụ thuộc vào tính chất của đường truyền dữ liệu vật lý). Gói được gửi chuyển tiếp qua mỗi nút trên mạng theo hình thức chuyển tiếp gói từ những nút khác hoặc bắt đầu từ chính nút đó. Một câu hỏi được đặt ra là: Đường kết nối nào sẽ được chọn để chuyển tiếp gói dựa theo địa chỉ đích của gói? Đây là chức năng dẫn đường của tầng mạng. Các chức năng khác như điều khiển lỗi và dòng cũng được thi hành ở tầng mạng, nhưng ở mức độ cao hơn giữa các nút và thông qua các nút trung gian. Hiện tượng nút nào đó luôn được ưu tiên chọn theo quyết định dẫn đường có khả năng xảy ra và nút như vậy sẽ trở thành cổ chai trong mạng. Điều khiển dòng nhằm giảm nhẹ vấn đề này được gọi là điều khiển tắc nghẽn. Quyết định dẫn đường có thể được tạo tại thời điểm khi kết nối được yêu cầu và được thiết lập. Quyết định dẫn đường có thể tạo ra trên cơ sở gói – tiếp - gói. Hình thức hóa bằng các giả thiết của kết nối. Chẳng hạn như đường truyền thông được coi như một chu trình ảo và có tính chất phân phát theo thứ tự các gói. Các gói được phát theo phương pháp sau được gọi chuyển mạch gói (datagram) và đòi hỏi thiết kế lại các gói theo dãy đúng đắn. Quyết định dẫn đường có thể tĩnh hoặc có thể được điều chỉnh theo trạng thái mạng. Tính toán quyết định dẫn đường có thể là tập trung hoặc phân tán trên một số nút cộng tác. Bài toán dẫn đường mạng được nghiên cứu rộng rãi.

Giữa mọi cặp nút (gửi và nhận), tồn tại một số đường để dẫn đường gói. Với một số ứng dụng phân tán, trông chờ vào cách thức dẫn đường phức. Nếu đường đi được chọn theo thuyết không quyết định đối với mỗi gói được chuyển tiếp, việc tải hệ thống được làm cân bằng hơn để ngăn ngừa hiện tượng thắt cổ chai. Lúc đố, các gói xuất hiện có thể không tuân theo dãy. Mặt khác, việc sao bội các gói sẽ được gửi có thể đưa đến yêu cầu tăng hiệu năng hệ thống. Nếu chỉ tiếp nhận gói được gửi thành công đầu tiên, có thể nhận thấy độ trễ liên nút ngắn hơn và xác suất thiếu gói giảm đi. Điều này lại đòi hỏi bổ sung cơ chế thiết kế lại gói và loại bỏ gói lặp lại. Các ứng dụng đòi hỏi hiệu năng cao và tin cậy cần có thể nâng cao tổng phí.

Hơn nữa, không phải tất cả các gói là gói dữ liệu. Sử dụng một số gói điều khiển cho giải pháp địa chỉ mạng và quảng bá trạng thái. Các giao thức này thường dùng dịch vụ chuyển mạch gói do các gói của chúng là nhỏ và ngắn hạn.

• ***Tầng giao vận:*** Theo quan điểm của HĐH, đây là tầng quan trọng nhất trong bộ giao thức bảy tầng vì nó ở vị trí trung gian giữa các mạng con truyền thông (phụ thuộc mạng: tầng vật lý, tầng kết nối dữ liệu và tầng mạng) với các tầng cao hơn - độc lập mạng (tầng phiên, tầng trình diễn và tầng ứng dụng). Trách nhiệm cơ bản của tầng giao

vận là cung cấp việc truyền thông hai mút tin cậy giữa các QT ngang hàng. Lỗi-vấn đề phụ thuộc mạng sẽ được che chắn khỏi các QT truyền thông. Các phiên truyền thông QT được đảm bảo bởi kết nối giao vận. Tầng giao vận tại nút gửi tách thông điệp thành các gói và chuyển chúng xuống tầng mạng để chuyển trên mạng. Tầng giao vận tại nút nhận thiết kế lại các gói thành thông điệp. Một số phiên nhỏ hơn có thể được kết hợp lại thành kết nối giao vận đơn nhằm đạt được sự tận dụng hiệu quả hơn kết nối, nếu chúng cùng đi tới một nút. Tương tự, tầng giao vận có thể chấp nhận một phiên chiếm giữ kết nối giao vận phức để làm tăng thông lượng phiên. Việc kết hợp và tập trung trong dịch vụ giao vận cần trong suốt đối với phiên. Phiên được phân lớp theo yêu cầu về kiểm soát lỗi và năng lực kết hợp. OSI định nghĩa 5 lớp (từ TP0 tới TP4) dịch vụ giao vận hỗ trợ phiên. Việc chọn lớp dịch vụ phụ thuộc vào yêu cầu của ứng dụng và chất lượng của mạng truyền thông hạ tầng. Dịch vụ truyền thông phổ dung nhất TP4 cho phép kết hợp các phiên, phát hiện lỗi và truyền lại. Nó là dịch vụ giao vận hướng kết nối tin cậy đối với mạng không tin cậy.

• ***Tầng phiên, trình diễn và ứng dụng:*** Các tầng trên tầng giao vận là không bản chất về mặt hệ thống và mạng truyền thông. Chúng là các dịch vụ bổ sung tới hệ thống.

Tầng phiên bổ sung dịch vụ hội thoại và đồng bộ cho tầng giao vận. Hội thoại làm thuận tiện việc thiết lập phiên còn đồng bộ cho phép các QT chèn thêm các điểm kiểm tra để khôi phục hiệu quả từ hiện tượng sụp đổ hệ thống.

Tầng trình diễn cung cấp mã hoá dữ liệu, nén và biến đổi mã đối với TĐ bằng các sơ đồ mã hóa khác nhau.

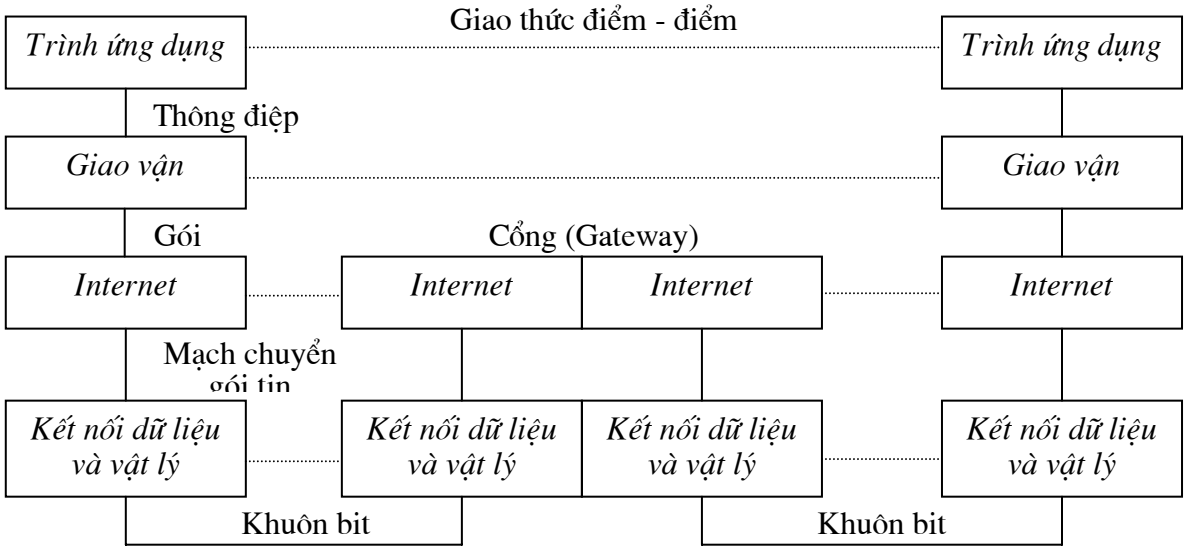
Chuẩn hóa tầng ứng dụng dành cho người thiết kế ứng dụng, chẳng hạn dịch vụ thư điện tử và truyền file.

2.5.2. Bộ giao thức TCP/IP

Đặc tả OSI cung cấp một diện mạo bề ngoài của truyền thông mạng. Khi đã cho một mạng vật lý hạ tầng, tồn tại hai kiểu tương tác hệ thống: truyền thông liên QT và truyền thông liên nút. Đặt ra hai câu hỏi đối với người thiết kế hệ thống: Làm thế nào để duy trì được truyền thông giữa cặp hai QT và việc dẫn đường TĐ theo các nút của mạng như thế nào ? Nói khác đi, tầng giao vận và tầng mạng là cốt yếu trong thiết kế hệ thống. TCP/IP là bộ giao thức tập trung vào hai vấn đề này trong môi trường liên mạng. TCP là giao thức tầng giao vận tương đương với TP4 trong bộ OSI. Đích cơ bản của bộ TCP/IP là mạng liên kết nối trong khi OSI là máy tính liên kết nối.

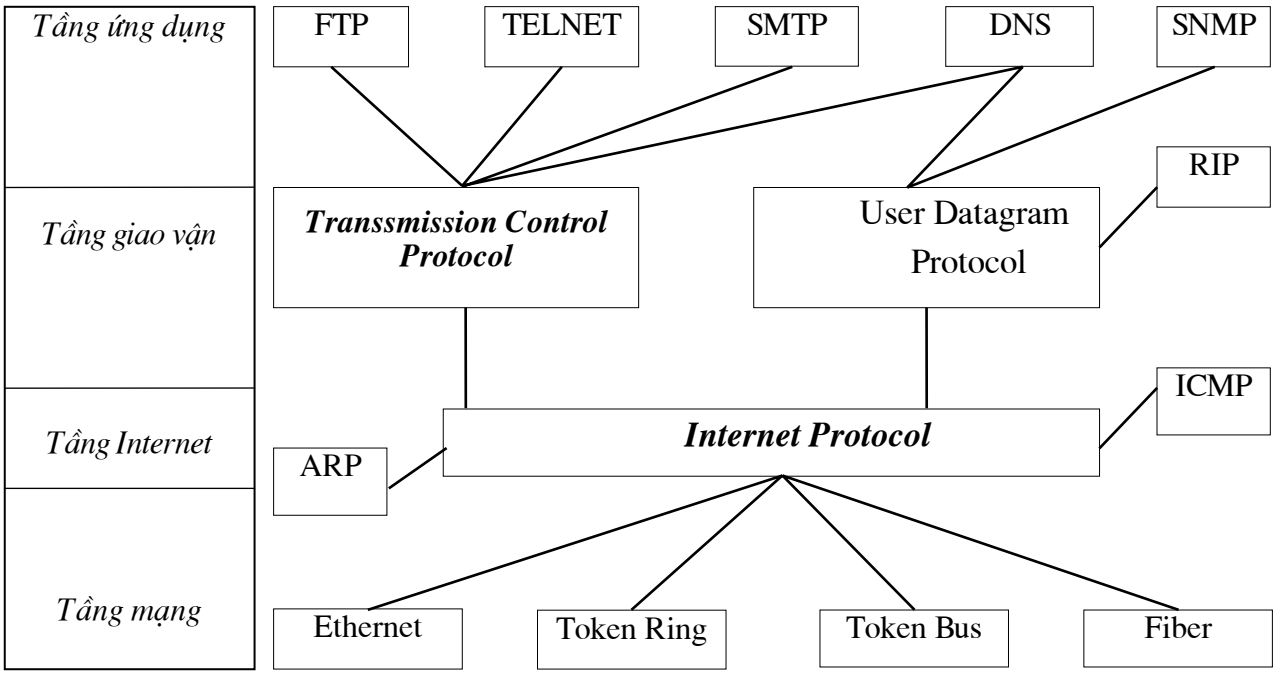
Hình 2.5 mô tả bộ giao thức TCP/IP cho hai mạng được kết nối qua một số cổng, còn hình 2.6 thể hiện các giao thức tương ứng với các tầng. Giao thức thực sự trong hình chỉ có hai tầng; các tầng khác được chỉ ra chỉ mang tính toàn vẹn. Tầng ứng dụng là không cần định rõ, thậm chí ngay cả tương tác giữa TCP và ứng dụng là không xác định để linh hoạt. Tầng kết nối dữ liệu và vật lý được chú ý như là một giao diện mạng mà hiện có rất nhiều chuẩn cho nó.

Mức giao vận có thể chọn hướng kết nối hoặc không kết nối. Dịch vụ này có thể được thực hiện bởi một chu trình ảo hoặc một gói tin tại tầng mạng. Như vậy, tổ hợp thành bốn cơ chế truyền thông QT. Truyền thông hướng kết nối, theo định nghĩa, có yêu cầu nghiêm ngặt đối với việc phân phát chính xác và đúng trình tự TĐ hơn so với truyền thông không kết nối. Nó là dịch vụ đúng đắn cho hầu hết ứng dụng. Hơn nữa, một gói tin (đơn vị truyền thông không kết nối) là dễ dàng và hiệu quả hơn khi thi hành trong tầng mạng, đặc biệt khi mạng hạ tầng là không tin cậy. Kết hợp tầng giao vận hướng kết nối và tầng mạng gói tin làm thích nghi một lớp rộng lớn các ứng dụng mạng. Đây là triết lý của TCP/IP (TCP hướng kết nối và IP mạch chuyển gói tin).



Hình 2.5. Bộ giao thức TCP/IP

Trách nhiệm duy trì truyền thông tin cậy từ mức mạng được chuyển tới mức HĐH, nơi cho phép điều khiển nhiều hơn so với tại mạng. Dữ liệu khác và TĐ điều khiển chỉ đòi hỏi dịch vụ không tin cậy, và điều đó được đáp ứng nhờ dịch vụ gói tin người dùng tại mức giao vận.

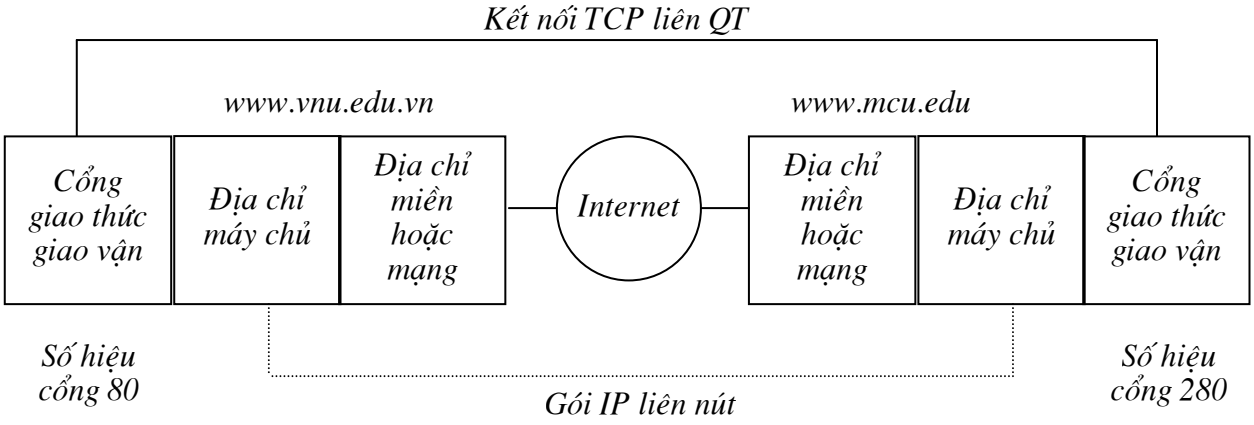


Hình 2.6. Các tầng trong bộ giao thức TCP/IP

Một QT có thể truyền thông trực tiếp tới QT ngang hàng nhờ sử dụng số hiệu QT (Pid) như địa chỉ nguồn và đích. Tuy nhiên, thuận tiện hơn dùng khái niệm cổng đối với nút truyền thông liên QT. Ví dụ một QT có thể dùng nhiều đường truyền thông tới một QT khác nhờ việc sử dụng cổng phức và các QT khác có thể chia xẻ cổng đích chung nhằm thực hiện kết nối đa điểm. Cổng được HĐH cục bộ khởi tạo và gán một số hiệu (id) tương ứng. Số hiệu cổng là duy nhất nội tại trong máy cục bộ. Nút mạng diện rộng tường minh đạt được nhờ việc ghép nối số hiệu cổng với máy chủ và địa chỉ mạng.

Hình 2.7 cho một cấu trúc địa chỉ IP và một kết nối TCP giữa hai nút. Nút kết nối được định danh bởi cặp địa chỉ máy chủ Internet và cổng giao vận.

Địa chỉ Internet đầy đủ chứa địa chỉ mạng và địa chỉ máy chủ nội tại trong mạng. Nếu mạng chứa mạng con, địa chỉ mạng (cũng được gọi là miền) được chia làm hai phần: địa chỉ mạng và địa chỉ mạng con. Một địa chỉ IP dài 32 bit. Các dạng địa chỉ IP được mô tả chi tiết ở phần sau.



Hình 2.7. Kết nối TCP và địa chỉ IP

Mức giao vận trong TCP/IP hoặc UDP/IP cung cấp các cổng đối với các dịch vụ hướng kết nối hoặc không kết nối. Giao diện chuyển mạch gói (socket) trong UNIX BSD IV được phát triển tại trường ĐHTH Berkeley là một ví dụ đối với cơ chế giao diện mà những ứng dụng chẳng hạn ống dẫn dòng UNIX cần được xây dựng. Socket là trừu tượng hóa vào-ra mạng cho phép thực hiện các thao tác đọc và ghi chuẩn. Lời gọi hệ thống ***socket*** tạo ra một socket, cho một đặc tả socket được dùng để đọc và ghi socket, tương tự như đặc tả file của một file được mở. Tham số trong lời gọi hệ thống ***socket*** mô tả họ giao thức và kiểu dịch vụ truyền thông sẽ được thiết đặt đối với socket. Với kết nối TCP, socket bắt buộc hướng tới cổng giao vận đích trước khi toán tử đọc hoặc ghi được giải thích. Lời gọi ***connect*** thực hiện việc làm phù hợp socket với cổng giao vận đích xa. HĐH tại đích xa cho số hiệu cổng tương ứng. Nếu số hiệu cổng cần thiết là đã biết thì nó có thể được QT đích gán nhờ lời gọi ***bind*** với chức năng ràng buộc socket tới một cổng cục bộ. Đối với truyền dữ liệu không kết nối, các lời gọi hệ thống ***sendio*** và ***revfrom*** được dùng. Yêu cầu về số hiệu socket và địa chỉ đích được mô tả như những tham số trong lời gọi. Không cần kết nối từ trước.

Do TCP/IP xác định trực tiếp hai tầng chủ yếu nhất để thiết kế hệ thống phân tán và mạng truyền thông, nó thường được dùng như là một mô hình để trình bày. Socket trong truyền thông không kết nối được sử dụng rộng rãi trong hệ phân tán và lập trình mạng.

Chức năng các giao thức trong bộ giao thức TCP/IP

Một số giao thức điển hình trong bộ giao thức TCP/IPđược liệt kê như dưới đây.

- **FTP** (File transfer Protocol): Giao thức truyền File lấy (gửi) File từ (tới) máy khác.
- **Telnet**: Chương trình mô phỏng thiết bị đầu cuối cho phép login vào máy chủ.
- **SMTP** (Simple Mail Transfer Protocol): Một giao thức thư tín điện tử.
- **DNS** (Domain Name phục vụ): Dịch vụ tên miền cho phép nhận ra máy tính từ một tên miền thay cho chuỗi địa chỉ Internet khó nhớ.

- **SNMP** (Simple Network Management Protocol): Giao thức cung cấp công cụ quản trị mạng.
- **RIP** (Routing Internet Protocol): Giao thức dẫn đường động.
- **ICMP** (Internet Control Message Protocol): Nghi thức thông báo lỗi.
- **UDP** (User Datagram Protocol): Giao thức truyền không kết nối cung cấp dịch vụ truyền không tin cậy nhưng tiết kiệm chi phí truyền.
- **TCP** (Transmission Control Protocol): Giao thức hướng kết nối cung cấp dịch vụ truyền thông tin cậy.
- **IP** (Internet Protocol): Giao thức Internet chuyển giao các gói tin qua các máy tính đến đích.
- **ARP** (Address Resolution Protocol): Cơ chế chuyển địa chỉ TCP/IP thành địa chỉ vật lý của các thiết bị mạng.

II.6. Kết quả thiết kế chủ yếu

Hệ phân tán bao gồm các QT đồng thời truy nhập tài nguyên phân tán (có thể được chia xẻ hoặc nhân bản) thông qua CTĐ trong môi trường mạng có thể không tin cậy và chứa các thành phần không cấu trúc. Một số vấn đề được đặt ra khi nghiên cứu thiết kế hệ phân tán. Thứ nhất, bằng cách nào các đối tượng trong hệ thống được mô hình hóa và định danh. Thứ hai, bằng cách nào kết hợp tương tác giữa các đối tượng và bằng cách nào chúng truyền thông cho nhau. Thứ ba là nếu đối tượng được chia xẻ hoặc nhân bản, bằng cách nào chúng được quản lý trong cấu thành được điều khiển. Thứ tư, sự an toàn của đối tượng và an ninh trong hệ thống bắt buộc phải xác định. Các vấn đề này được tóm lược trong phần tiếp theo và được phân tích kỹ trong các chương sau.

II.6.1. Mô hình đối tượng và sơ đồ tên

Đối tượng trong hệ thống máy tính là QT, file dữ liệu, bộ nhớ, thiết bị và mạng. Theo truyền thống, các kiểu đối tượng khác nhau được thi hành khác nhau. Theo đúng tinh thần của tính trong suốt, giả thiết rằng mọi đối tượng được trình bày theo cùng một cách thức. Do các đối tượng mang nghĩa đầy đủ chỉ khi chúng được truy nhập vì vậy mỗi đối tượng cần được phù hợp với các thao tác hoàn toàn xác định để truy nhập đối tượng. Bởi vậy, mỗi đối tượng được trừu tượng hóa bằng tập các thao tác truy nhập được đối với nó. Chi tiết vật lý của đối tượng là trong suốt đối với các đối tượng khác. QT thực hiện công việc quản lý đối tượng trở thành phục vụ đối tượng. Nói khác đi, đối tượng được cô lập qua các phục vụ và chỉ còn các phục vụ là các thực thể nhìn thấy được trong hệ thống. Như vậy, trong hệ thống có phục vụ QT, phục vụ File, phục vụ bộ nhớ... Khách là một phục vụ rỗng, nó truy nhập các phục vụ đối tượng.

Để giao thiệp được với một phục vụ thì phục vụ đó phải được định danh. Tồn tại chỉ ba cách định danh phục vụ:

- Định danh bằng tên,
- Định danh bằng hoặc địa chỉ vật lý hoặc địa chỉ lôgic
- Định danh bằng dịch vụ mà phục vụ đó cung cấp,

Tên được dùng cho mục đích định danh, được giả thiết tổng quát là duy nhất trong khi đó cho phép tồn tại tính đa địa chỉ tương ứng với cùng một phục vụ, địa chỉ có thể thay đổi khi phục vụ chuyển động. Tên là trực giác và trong suốt hơn so với địa chỉ nhưng địa chỉ chứa thông tin cấu trúc để định vị phục vụ. Giải pháp ánh xạ tên tới địa chỉ lôgic được thực thi bởi phục vụ tên trong HĐH. Giải pháp địa chỉ ánh xạ địa chỉ lôgic

thành địa chỉ vật lý là chức năng của dịch vụ mạng. Trong nhiều hệ thống, *cổng* (port) được coi là một địa chỉ lôgic. Tương ứng nhiều cổng tới một dịch vụ theo các đa điểm vào tới phục vụ. Mặt khác, đa phục vụ có thể chia xẻ cùng một cổng. Tiếp cận này được đặt ra theo định danh tên trong hệ phân tán. Phương pháp thứ ba - định danh theo dịch vụ là độc đáo hơn. Khách chỉ quan tâm đến dịch vụ được yêu cầu còn ai cung cấp dịch vụ này thì không liên quan. Điều hợp lý là đa phục vụ có thể cung cấp cùng một dịch vụ. Tiếp cận này cũng được dùng cho thi hành HĐH tự trị cộng tác. Một cách tự nhiên, cần thiết một giao thức giải pháp để chuyển từ dịch vụ tới phục vụ.

Mô hình đối tượng và định danh theo tên là những vấn đề cơ sở buộc được quyết định ngay khi thiết kế HĐH. Cấu trúc hệ thống, quản lý không gian tên, giải pháp tên và phương pháp truy nhập, tất cả phụ thuộc vào sơ đồ tên của các đối tượng trong hệ thống.

2.6.2. Công tác phân tán

Các QT đồng thời tương tác nhau đòi hỏi cộng tác để đồng bộ hoạt động. Nói chung có 3 kiểu thủ tục đồng bộ:

- *Đồng bô bằng ngăn cấm*: Tập QT (hoặc sự kiện) bắt buộc đi tới một điểm đồng bộ chung trước khi chúng có thể tiếp tục,

- *Cộng tác theo điều kiện*: Một QT (hay sự kiện) bắt buộc phải chờ một điều kiện được khởi tạo một cách dị bộ từ các QT tương tác nhằm duy trì thứ tự thực hiện nào đó.

- *Loại trừ ràng buộc*: Các QT đồng thời bắt buộc loại trừ ràng buộc khi truy nhập tài nguyên được chia xẻ tới hạn.

Ngầm hiểu rằng đồng bộ cần tới tri thức liên quan đến thông tin trạng thái về các QT khác. Thông tin trạng thái đầy đủ là điều rất khó khăn vì trong hệ phân tán không có bộ nhớ chia xẻ. Thông tin trạng thái có thể được chuyển tới bằng việc gửi TĐ. Tuy nhiên, thời điểm TĐ được nhận là không đúng và không đầy đủ do chuyển TĐ bị trễ trên mạng. Quyết định một QT được tiếp tục hay không buộc phải dựa vào các giao thức giải pháp phân tán dựa trên TĐ. Một giải pháp thường đi liền tới vấn đề cộng tác phân tán là giao cho một bộ cộng tác tập trung chọn lựa trách nhiệm cộng tác. Vai trò của bộ cộng tác tập trung có thể được chuyển từ QT này sang QT khác nhằm làm cho bộ cộng tác không trở thành điểm trung tâm của lỗi. Bằng cách áp dụng cách thức này, buộc tồn tại các điều khoản cho phép một QT được chọn trở thành bộ cộng tác mới nếu như bộ cộng tác chính thức có lỗi hoặc có quyết định làm nó từ bỏ trách nhiệm của mình.

Một vấn đề khác liên quan mật thiết với tính đồng bộ là sự bế tắc của các QT. Tương tự như vấn đề bế tắc trong hệ điều hành tập trung, các QT đồng thời có thể chạy đúng đắn, không xâm phạm ràng buộc của bất cứ đồng bộ nào song lại có thể dẫn đến hiện tượng bế tắc do gặp phải chu trình chờ lẫn nhau. QT và tài nguyên trong hệ phân tán là rất hỗn tạp. Để phòng tránh và ngăn ngừa bế tắc, đôi khi là không thực tế khi tiến hành việc ngăn cản bế tắc hoặc dùng chiến lược phòng tránh để kiểm soát bế tắc. Chúng ta xem xét khả năng phát hiện bế tắc và thử khôi phục lại nếu có thể. Trong hệ phân tán, vấn đề phát hiện bế tắc lại là vấn đề không tầm thường vì không có được thông tin trạng thái toàn cục của hệ phân tán. Tồn tại một số câu hỏi như ai là người đề xướng thuật toán phát hiện bế tắc, làm thế nào thuật toán được thi hành theo hình thức CTĐ của hệ phân tán, QT nào sẽ là nạn nhân để thoát ra/ giải quyết bế tắc và bằng cách nào nạn nhân được khôi phục. Hiệu lực của giải pháp bế tắc và chiến lược khôi phục được coi là quan trọng hơn so với phát hiện bế tắc trong hệ phân tán.

Giải pháp phân tán cho vấn đề đồng bộ và bế tắc là cố gắng đối sánh từng phần thông tin trạng thái tổng thể và sử dụng nó để ra quyết định. Nhiều ứng dụng không đòi hỏi thông tin trạng thái tổng thể tuyệt đối mà tiến hành giống như các QT phức tạp thỏa thuận theo am hiểu của chúng về hệ thống. Giao thức chấp nhận là thuật toán CTĐ cho phép hành động nhất trí trong hệ phân tán cho phép có thể có lỗi thành phần chấp nhận được. Việc đưa ra sự nhất trí đòi hỏi chuyển đổi các tri thức cục bộ xuyên theo các phía cộng tác. Đây là bài toán không quá khó ngoại trừ việc bộ xử lý nào đó ngả về lỗi hoặc không đáng tin cậy. Tuy nhiên, trực giác có nhận thể thấy rằng hoạt động chấp nhận là không thể làm được trong hệ phân tán dị bộ.

Đồng bộ phân tán và nắm giữ bế tắc là hai công cụ cộng tác QT chính yếu để xây dựng dịch vụ phân tán.

2.6.3. Truyền thông liên QT

Truyền thông liên QT là vấn đề bản chất nhất trong mọi thiết kế hệ phân tán do mọi thứ đều dựa vào nó. Trong HĐH, sự tương tác giữa các QT và dòng thông tin giữa các đối tượng phụ thuộc vào truyền thông. Ở mức thấp nhất, CTĐ mang nghĩa của truyền thông trong hệ phân tán. Truyền thông liên QT có thể được thực hiện bởi dịch vụ nguyên thủy CTĐ đơn giản. Tuy nhiên, do yêu cầu về tính trong suốt trong truyền thông nên đòi hỏi cung cấp những phương pháp truyền thông lôgic mức cao nhằm che dấu đi chi tiết vật lý của CTĐ.

Hai khái niệm quan trọng được dùng để thực hiện mục tiêu này là mô hình Client/Server và RPC. Mô hình Client/Server là mô hình lập trình nhằm cấu trúc các QT trong hệ phân tán. Trong mô hình này, mọi liên tương tác hệ thống được xem như một cặp trao đổi thông điệp trong đó QT khách gửi TĐ hỏi tới phục vụ và chờ phục vụ đáp lại một TĐ trả lời. CTĐ hỏi/đáp như vậy tương tự như khái niệm lời gọi thủ tục trong ngôn ngữ lập trình trong đó thủ tục gọi truyền tham số cho thủ tục được gọi, và thủ tục được gọi trả lại kết quả cho thủ tục gọi để hoàn thành thực hiện thao tác. Trao đổi thông tin hỏi/đáp giữa khách và phục vụ có thể được trình bày như lời gọi thủ tục tới một phục vụ từ xa. Truyền thông RPC được xây dựng dựa trên mô hình Client/Server và CTĐ đã được đề xuất như cơ chế truyền thông liên QT đối với mọi hệ phân tán trong tương lai.

Bàn luận trên đây về CTĐ, mô hình Client/Server và RPC với giả thiết truyền thông điểm-điểm. Khái niệm *nhóm* là bản chất trong phần mềm phân tán. Cộng tác QT trong hoạt động của nhóm. Quản lý nhóm hay truyền thông nhóm (tán phát bội - multicast hoặc quảng bá - broadcast) là cần thiết. Nhu cầu truyền thông đi qua một số tầng giao thức và truyền tới một số nút phân tán vật lý. Điều đó sẽ nhạy cảm hơn tới lỗi trong hệ thống. Truyền thông nhóm tin cậy và nguyên tử vẫn là vấn đề mở trong hệ phân tán. Hơn nữa, quản trị nhóm, nền tảng của phương thức làm việc cộng tác có hỗ trợ của máy tính (CSCW), vẫn chưa đạt được độ thuần thục do thiếu kinh nghiệm trong các ứng dụng phần mềm phân tán. Vấn đề truyền thông liên QT được khảo sát tỷ mỷ trong chương 4.

2.6.4. Tài nguyên phân tán

Về mặt logic, tài nguyên cần cho tính toán chỉ là dữ liệu và năng lực xử lý. Về mặt vật lý, dữ liệu nằm trong bộ nhớ hoặc bộ nhớ phụ phân tán (dạng các file). Năng lực xử lý được tính gộp từ năng lực xử lý của tất cả bộ xử lý. Mục tiêu nền tảng của xử lý phân tán là đạt được tính trong suốt khi định vị năng lực xử lý tới QT, hoặc ngược lại, phân tán các QT (hoặc tải) tới các bộ xử lý. Phân tán tải tĩnh trong hệ phân tán được chỉ dẫn như lập lịch đa xử lý. Mục tiêu là tối thiểu thời gian hoàn thành tập các QT. Vấn đề chính là tác động của tổng phí truyền thông khi thiết kế chiến lược lập lịch. Nếu phân

bố tải (hay phân bố lại tải) là được làm theo cách động, nó được gọi là chia xẻ tải. Mục tiêu là cực đại sự tận dụng tập các bộ xử lý. Vấn đề nguyên thuỷ là di trú QT cùng chiến lược và cơ chế cho nó. Ngoài những yêu cầu về hiệu năng tốt hơn và tận dụng cao hơn, nhiều ứng dụng phân tán được ràng buộc về thời gian. Trong những trường hợp đó, lập lịch QT phải đáp ứng một số yêu cầu về thời gian thực. Lập lịch phân bố tải tĩnh và động và lập lịch thời gian thực được trình bày trong chương 5.

Tính trong suốt áp dụng trong dữ liệu phân tán là vấn đề phức tạp hơn. Nếu file là thực thể dữ liệu cơ bản thì một hệ thống file trong suốt có nghĩa là nó đưa ra một cách nhìn hệ thống file đơn đối với các file rải rác trong một môi trường phân tán. Hệ thống file phân tán là hệ thống file trong suốt. Có thể mở rộng khái niệm dữ liệu trong suốt thêm một chút nữa với giả thiết rằng khối dữ liệu là nhỏ hơn file và dữ liệu nằm trong các môđun bộ nhớ phân tán. Một hệ thống bộ nhớ trong suốt cho phép trình diễn một cái nhìn bộ nhớ chia xẻ đơn đối với các bộ nhớ phân tán vật lý. Điều cốt lõi là mô phỏng một hệ thống bộ nhớ chia xẻ, được gọi là bộ nhớ chia xẻ phân tán (distributed shared memory) nếu tổng phí truyền thông là tha thứ được.

Vấn đề chung cho cả hệ thống file phân tán và bộ nhớ chia xẻ phân tán là chia xẻ và nhân bản dữ liệu. Cả hai vấn đề chung này yêu cầu các giao thức duy trì tính nhất quán và toàn vẹn dữ liệu và như vậy, kết quả của chia xẻ và sự tồn tại nhân bản là trong suốt đối với người dùng. Mặc dù, hai vấn đề này tương đương lôgic song vẫn tồn tại những sự khác biệt tinh tế trong khi thi hành giữa hệ thống file phân tán và bộ nhớ chia xẻ phân tán. Vấn đề hệ thống file phân tán dựa trên quan điểm của file, trong khi đó trong hệ thống bộ nhớ chia xẻ phân tán lại nhấn mạnh hơn vào mức độ am hiểu của hệ thống đối với QT.

2.6.5. Thứ lỗi và an toàn

Do tính mở trong môi trường điều hành, hệ phân tán dễ bị tấn công bởi hỏng hóc và de doạ an toàn. Cả hai (lỗi hỏng hóc và đe dọa an toàn) được coi là lỗi hệ thống. Hỏng hóc là lỗi do chỉ thị không định trước (vô ý) và vi phạm an toàn là lỗi do chỉ thị chủ định (cố ý). Hệ phân tán tin cậy là hệ thống có tính thứ lỗi theo nghĩa trong hệ thống đó có những cơ chế và giải pháp đối với hai loại lỗi trên đây.

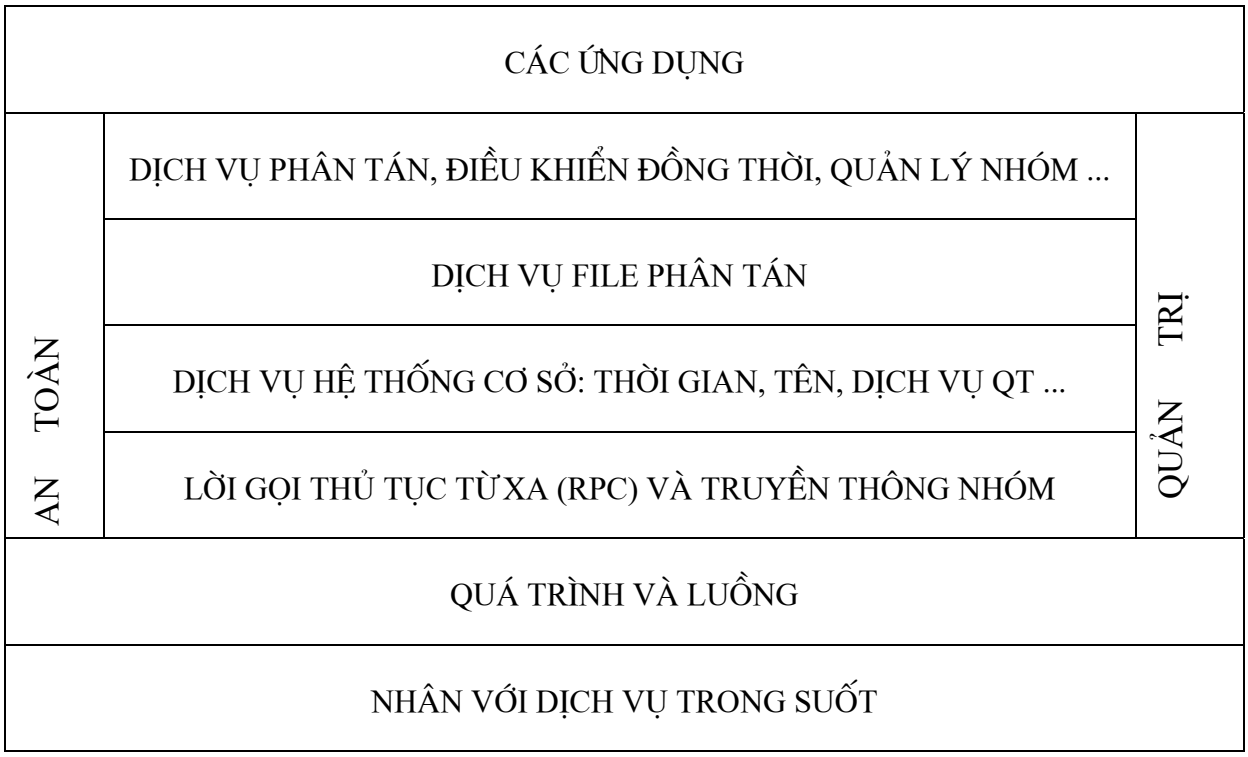
Vấn đề hỏng hóc có thể được giảm nhẹ nếu trong hệ thống phân tán tồn tại sự dư dật. Dư dật là tính chất vốn có gắn liền với hệ phân tán do dữ liệu và tài nguyên có thể được nhân bản. Thêm vào đó, thông thường việc khôi phục do lỗi hỏng hóc yêu cầu việc chạy lại QT bị lỗi và các QT khác nếu có dính dáng đến hỏng hóc. Thông tin trạng thái thực hiện bắt buộc phải bảo quản để khôi phục chạy lại mà đây lại là một vấn đề khó khăn trong hệ phân tán. Thông thường, sử dụng giải pháp điểm kiểm tra cho phép hỗ trợ chạy lại QT và khôi phục.

Mối quan tâm về an toàn ngày càng tăng nhanh trong mạng và hệ phân tán. Theo quan điểm của HĐH, cần quan tâm tới tính tin cậy của QT truyền thông và tính tin cẩn và toàn vẹn dữ liệu. Vấn đề xác thực và giấy phép đảm nhận chất lượng về tính duy nhất trong hệ phân tán. Về vấn đề xác thực, *khách* và cũng vậy *phục vụ* và *thông điệp* bắt buộc phải được xác thực. Với vấn đề giấy phép, điều khiển truy nhập phải đủ năng lực xuyên qua mạng vật lý với các thành phần hỗn tạp theo các đơn vị quản trị khác nhau sử dụng các mô hình khác nhau.

II.7. Môi trường tính toán phân tán

Hình 2.8 mô tả môi trường tính toán phân tán (Distributed Computing Enviroment - DCE). Mô hình này có biến đổi đôi chút so với kiến trúc DCE được Tổ chức phần mềm mở (Open Software Foudation: OSF) đề xuất. OSF là một dự án liên kết của nhiều

công ty máy tính Mỹ với mục đích phát triển và chuẩn hóa một môi trường UNIX mở mà miễn phí không bị ảnh hưởng bởi AT&T và Sun. So với hầu hết các thi hành UNIX là sẵn sàng mở rộng hợp nhất để hỗ trợ tính toán phân tán, sản phẩm chính từ OSF là DCE, một bó tích hợp phần mềm và tool nhằm phát triển ứng dụng phân tán trên các HĐH đã có. DCE cung cấp nhiều dịch vụ thiết yếu, chẳng hạn như các dịch vụ luồng (thread), RPC, an toàn, và thư mục sẵn sàng tích hợp vào nhiều nền UNIX và không UNIX. Việc sắp đặt mỗi dịch vụ riêng trong kiến trúc phân cấp là quan trọng. Tại trung tâm của hệ thống là dịch vụ nhân và dịch vụ giao vận, là giao diện cung cấp các dịch vụ truyền thông tới các QT tại các host khác. QT và luồng là những đơn vị tính toán cơ sở được nhân hỗ trợ. Luồng là dạng đặc biệt của QT có khả năng thi hành hiệu quả phục vụ đồng thời. Mọi dịch vụ khác nằm trong không gian người dùng và tương tác lẫn nhau theo nghĩa của RPC và truyền thông nhóm. Các dịch vụ thời gian, tên, và QT là những ví dụ của dịch vụ hệ thống cơ sở. Với file như là đối tượng nguyên thủy trong hệ thống, các dịch vụmức cao, chẳng hạn như điều khiển đồng thời và quản lý nhóm cần được xây dựng dựa trên dịch vụ file phân tán. Cuối cùng, chức năng an toàn và quản trị cần được tích hợp từ mọi tầng.



Hình 2.8. Kiến trúc môi trường tính toán phân tán

CÂU HỎI VÀ BÀI TẬP

2.1. Đặc điểm của hệ điều hành phân tán. Mục tiêu thiết kế hệ điều hành phân tán.

2.2. Tính trong suốt trong hệ điều hành phân tán: khái niệm và các thể hiện của nó.

2.3. Các mức dịch vụ trong hệ phân tán.

2.4. Sơ bộ về các kết quả thiết kế chủ yếu: mô hình đối tượng - tên, cộng tác QT, truyền thông liên QT, tài nguyên phân tán, thứ lỗi - an toàn

CHƯƠNG III. QUÁ TRÌNH ĐỒNG THỜI VÀ LẬP TRÌNH

Trong HĐH phân tán, hai phần tử thiết yếu là QT và luồng (thread). Quản lý QT được phân lớp triển khai theo ba khu vực (cũng là ba chức năng liên quan đến quản lý QT trong hệ phân tán):

+ Truyền thông QT,

+ Đồng bộ hoá QT,

+ Lập lịch QT.

Ba chức năng này thuộc vào một thể thống nhất và không tách rời nhau.

Các chức năng truyền thông và đồng bộ có mối quan hệ mật thiết cả về khái niệm và lẫn khi thi hành. Các khái niệm và việc thi hành phối hợp được trình bày trong hai chương III và IV.

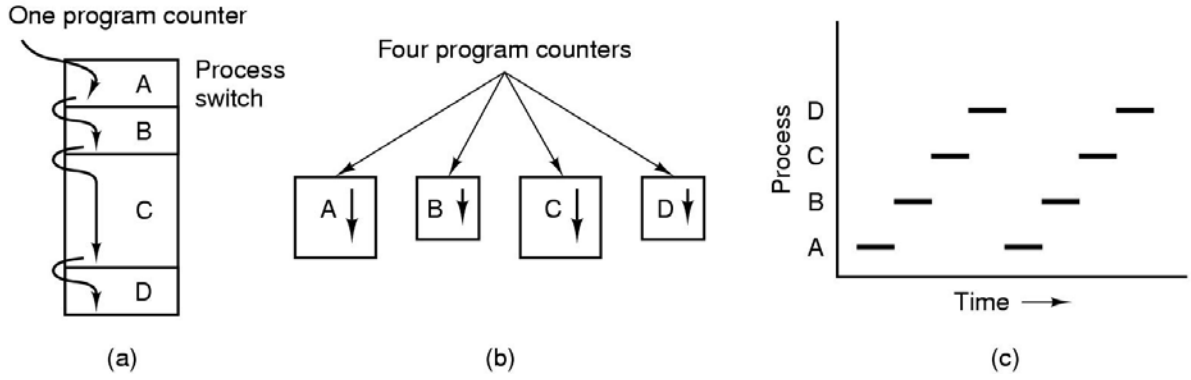
Lập lịch QT liên quan đến trình tự thực hiện các QT để đạt được hiệu suất tốt nhất cho hệ thống. Trình tự thực hiện QT tuỳ thuộc vào đồng bộ QT trong khi hiệu suất lại phụ thuộc vào năng lực lớn mạnh của kĩ thuật truyền tin cơ sở và thời gian trễ trong quá trình truyền tin. Do đặc thù khá riêng biệt nên lập lịch QT được trình bày trong chương V. Dù cho truyền thông QT, đồng bộ QT và lập lịch QT có những đặc điểm chung như trong HĐH tập trung, song nhằm mục đích định hướng hệ phân tán cho nên trình bày quản lí QT có trong ba chương III, IV và V .

Trước hết bắt đầu với các định nghĩa và các đặc điểm của điều khiển QT.

3.1. Khái niệm QT và luồng

QT là đối tượng trong HĐH, biểu thị việc thực hiện một chương trình trong một phiên làm việc: QT là một đơn vị tính toán cơ sở trong hệ thống.

Một số điểm phân biệt hai khái niệm chương trình và QT: Chương trình liên quan đến bài toán cần giải quyết (các tham số hình thức), tên chương trình, độ dài, ngôn ngữ nguồn …. QT là một lần sử dụng chương trình đã có để giải quyết bài toán trong một tình huống cụ thể (tham số đã được cụ thể). QT có trạng thái quá trình, bao gồm trạng thái phân bố các thành phần của QT trong bộ nhớ trong ...



Đa chương trình với 4 chương trình

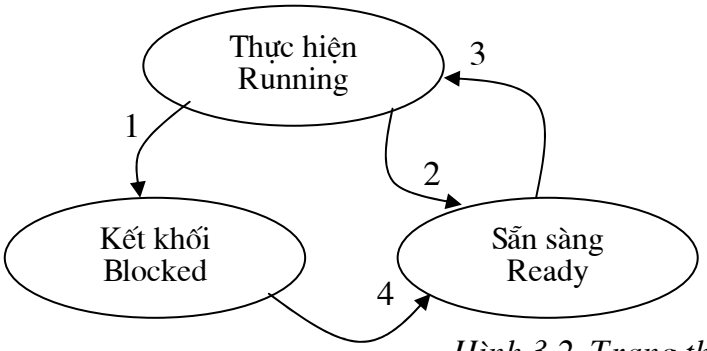
Mô hình quan niệm của 4 QT tuần tự

Chỉ 1 chương trình hoạt động trong mọi thể hiện

Hình 3.1. Quá trình

QT được gọi là *đơn* nếu các lệnh (thành phần con) trong nó được thực hiện một cách tuần tự. Thuật ngữ *đồng thời* liên quan đến việc mô tả sự thực hiện đồng thời các QT

đơn. Giữa hai QT có những thành phần được phép thực hiện đồng thời. Các thành phần khác cần được đồng bộ hoặc truyền thông giữa chúng.



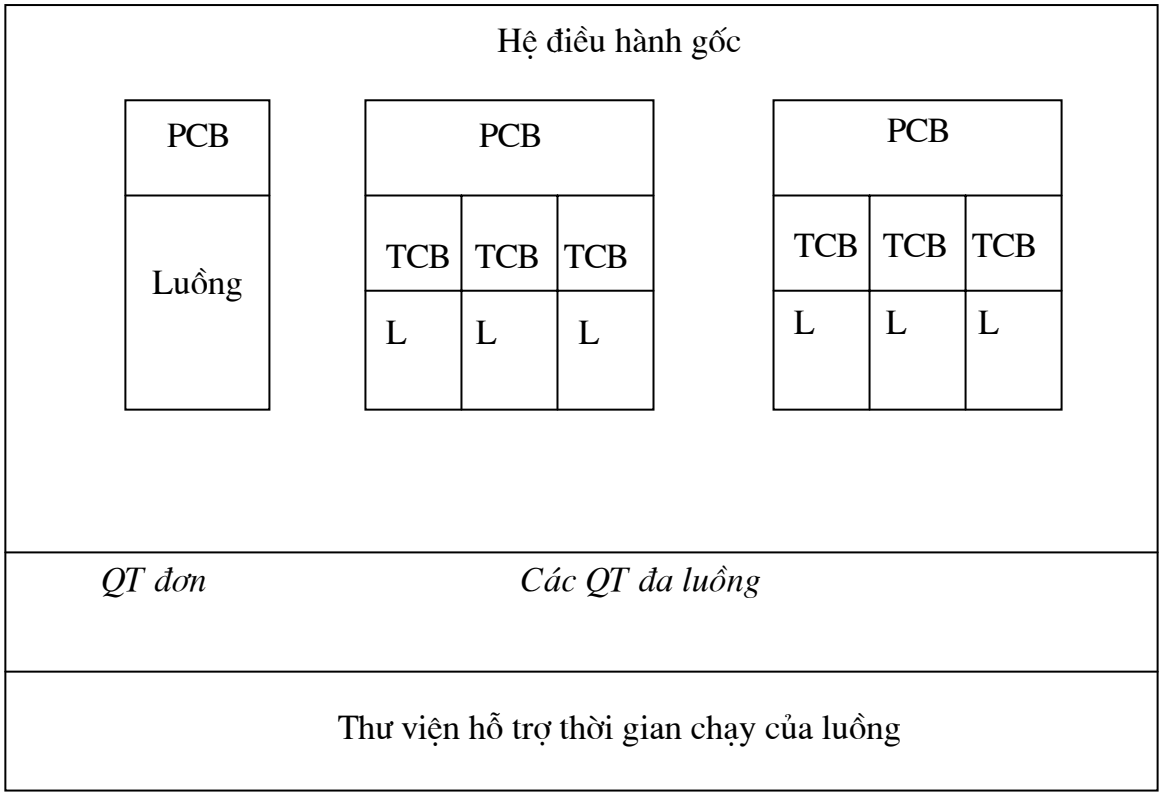
1. QT kết khối để nhập dữ liệu
2. Bộ lập lịch chọn QT khác
3. QT được chọn
4. Nhập dữ liệu xong

Hình 3.2. Trạng thái của QT

Luồng (thread) là một biến thể của QT, tương ứng với trường hợp khi thực hiện một QT lại sinh ra một QT khác. QT đơn thực chất là QT đơn luồng, trong thời gian thực hiện, nó không tạo ra một QT mới.

QT đa luồng là QT mà trong khi thực hiện nó cho ra một QT mới: Đây là trường hợp đặc biệt của tính đồng thời khi QT cha và QT con "đồng thời" thực hiện và chia xẻ tài nguyên CPU cùng bộ nhớ trong và mỗi luồng có trạng thái riêng của mình.

Hình 3.3 trình bày tính đồng thời hai mức của một QT và một luồng. Tại mức thứ nhất (mức thấp), các QT chạy đồng thời dị bộ theo HĐH gốc. Khi xem xét tại mức đồng thời thứ hai, mỗi QT đồng thời chạy tựa một máy tính ảo hỗ trợ tính đồng thời của các luồng. Một QT được hiểu một cách đơn giản như một không gian địa chỉ lôgic mà tại đó luồng thực hiện.



Hình 3.3. Quá trình và luồng

Trong HĐH tập trung, thuật ngữ "khối điều khiển QT" PCB (Process Control Block) dùng để chỉ cấu trúc dữ liệu chứa các thông tin về QT, hỗ trợ việc điều khiển CPU và

điều khiển QT (thông tin trạng thái: thanh ghi địa chỉ lệnh, nội dung các thanh ghi, con trỏ stack, cổng truyền thông, và đặc tả file ...). PCB chứa các thông tin cần thiết để luân chuyển thực hiện QT được quản lý bằng HĐH gốc.

Giống như QT, luồng cũng có các thủ tục và stack riêng. Thông tin trạng thái về luồng được cho trong khối điều khiển luồng TCB (Thread Control Block). TCB được quản lý bởi Thư viện hỗ trợ thời gian chạy luồng. TCB là riêng đối với mỗi luồng trong khi PCB lại được chia xẻ cho các luồng tương tác để đồng bộ và truyền thông. Thông tin trong TCB ít hơn nhiều so với thông tin trong PCB và chỉ gồm nội dung các thanh ghi (bộ đếm chương trình, đỉnh stack, tập thanh ghi). Các thông tin trạng thái khác được bảo quản trong PCB. Bởi lý do đó, luồng được gọi *QT nhẹ* trong khi đó QT được gọi là *QT nặng*. Chú ý về mặt sử dụng ký hiệu, trong các HĐH hoạt động theo chế độ mẻ (batch, lô) dùng ký hiệu TCB với nghĩa khác là "khối điều khiển bài toán" - Task Control Block.

Thi hành luồng trong không gian người dùng được chỉ ra trong hình 3.4, và chỉ có QT (mà không phải là luồng) được HĐH nhìn thấy. Ngoài ra, luồng được thực hiện trong không gian nhân và được quản lý trực tiếp bởi HĐH gốc. Luồng trong một QT được khởi tạo tĩnh hoặc động bởi một QT điều khiển hoặc một luồng khác.

3.1.1 Các ứng dụng luồng

Luồng có nhiều ứng dụng trong HĐH phân tán. Chúng thường được dùng khi thi hành một QT phục vụ cung cấp các dịch vụ tương tự hoặc có quan hệ tới các QT đa khách, chẳng hạn như phục vụ trạm cuối hoặc phục vụ file.

Khi một yêu cầu phục vụ (serving request) từ QT khách tới một QT phục vụ đơn luồng, thì QT phục vụ này tự tạm ngừng (có thể quan niệm phục vụ này như một tài nguyên được điều khiển bởi một semaphore nhị phân) để chờ đợi hoàn thiện các điều kiện hoặc thao tác nào đó từ trước. Tuy nhiên, việc tạm ngưng phục vụ lại kết khối các yêu cầu khách mới được đưa tới phục vụ. Để tăng thông lượng hệ thống, đa bản sao của cùng một phục vụ được khởi tạo cho các yêu cầu khác nhau tới cùng một phục vụ một cách đồng thời. Do các luồng phục vụ này có mã lệnh tương tự nhau và bắt buộc phải tương tác nhau qua thông tin toàn cục được chia xẻ, vì vậy chúng được nhóm trong một không gian địa chỉ, và như vậy là đã khởi tạo tính đa luồng trong một phục vụ đơn.

QT khách cũng được điều khiển theo cách hoàn toàn tương tự. Một QT khách yêu cầu tạo ra nhu cầu đồng thời tới các phục vụ và bỏ qua việc bị kết khối bởi bất kỳ từ các yêu cầu dịch vụ này.

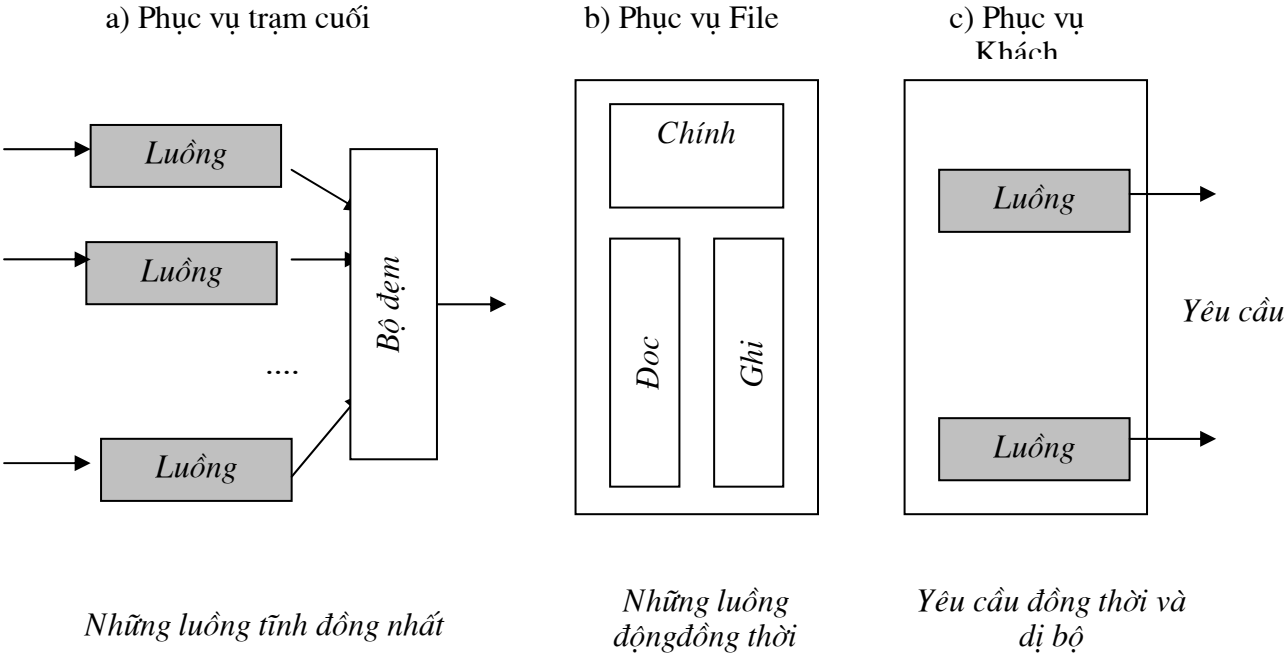
Hình 3.4 mô tả ba ứng dụng luồng trong hệ phân tán.

• Phục vụ trạm cuối (phức hợp và tập trung dữ liệu) trong hình 3.4a:

Chức năng tập hợp dữ liệu đa thành phần từ nhiều trạm cuối vào bộ đệm chung và gửi dữ liệu phức trong một bộ đệm chung tới một máy tính (hay mạng). Nếu không dùng đa luồng, phục vụ trạm cuối cần bầu cử trạm cuối đưa vào bằng cách sử dụng dịch vụ nguyên thuỷ không kết khối. Theo quan niệm, sẽ đơn giản hơn nếu thiết kế phục vụ thành đa luồng, mỗi từ chúng đáp ứng một input riêng. Mã lệnh của các luồng này là đồng nhất nên được chia xẻ như *mã thực hiện lại* mà mỗi luồng có stack cục bộ riêng của mình. Việc truy nhập vào buffer cùng được chia xẻ bởi các luồng cần loại trừ ràng buộc. Việc loại trừ ràng buộc có thể đạt được bằng cách sử dụng phương pháp đồng bộ bộ nhớ chia xẻ như semaphore hay bộ kiểm tra, vì tất cả các luồng chia xẻ một vùng địa chỉ. Hơn nữa đồng bộ luồng có thể hiệu quả hơn nhiều vì QT đồng bộ chỉ gọi phần cục bộ và có thể tránh được việc gọi nhân trong một số trường hợp. Các luồng trong ví

dụ phục vụ trạm cuối này được tạo tĩnh và chạy không tiền định. Về cơ bản, chúng chạy giống như một bộ điều khiển ngắt thật sự.

• Hình 3.4b, trình bày một tình huống ứng dụng luồng khác. Phục vụ File thi hành các thao tác dịch vụ file khác nhau theo yêu cầu từ khách. Một luồng được tạo ra cho mỗi thao tác và điều khiển được quay lại luồng chính, và như vậy luồng chính có thể tiếp nhận một yêu cầu mới. Trong những điều kiện nào đó, luồng được kết khối, và một luồng khác được lập lịch để thực hiện. Luồng ngừng tồn tại khi công việc của nó hoàn thành. Kết khối luồng và lập lịch luồng được trình bày ở phần sau. Chú ý là trong ví dụ về phục vụ file, tồn tại luồng chính phục vụ như một trình điều phối công việc cho các dịch vụ file đồng thời; việc khởi tạo và kết thúc luồng là động. Tạo luồng và huỷ luồng là đơn giản vì lí do dùng lại không gian nhớ. Đây là cấu trúc luồng phổ biến cho phần lớn các loại phục vụ này.



Hình 3.4. Các ứng dụng luồng

• Ví dụ thứ 3 về luồng cho trong hình 3.4(c) là một khách đưa ra nhiều yêu cầu tới các phục vụ khác nhau. Đa luồng trong QT khách làm cho nó có thể đạt được đồng thời các dịch vụ và dị bộ thậm chí khi thông tin yêu cầu trả lời là đồng bộ. Một ứng dụng hữu dụng của cấu trúc này là việc cập nhật đồng thời các bản sao file nhân bản mà được quản lí bởi nhiều phục vụ file.

Hỗ trợ luồng trong nhiều HĐH hiện đại rất rộng lớn bởi vậy người lập trình ứng dụng có thể viết các chương trình đồng thời một cách hiệu quả. Ví dụ, ứng dụng duyệt web đa luồng có thể khởi tạo việc truyền fie đa thành phần, cho phép làm chậm QT truyền file ở Internet chồng lên nhau.

• Một ví dụ ứng dụng đa luồng khác là hệ thống đa cửa sổ. Các ứng dụng toạ độ cửa sổ trở nên dễ dàng hơn nếu chúng được thực hiện ở trong luồng với chia xẻ vùng địa chỉ logic chia xẻ. Chẳng hạn, một luồng có thể thực hiện một hoạt động trong cửa sổ này mà kết quả lại để trong một cửa sổ khác. Để thực hiện hiệu quả những ứng dụng này, các luồng ưu tiên và các hỗ trợ bộ đa xử lí được đòi hỏi.

3.1.2 Thi hành luồng trong không gian người dùng

Hỗ trợ luồng như một gói thêm vào đã được thực hiện trên nhiều hệ thống bao gồm gói luồng DCE từ Tổ chức phần mềm mã mở OSF và gói QT nhẹ (LWP: Light Weight Process) từ Sun. Vấn đề thi hành cốt lõi là nắm giữ được các lời gọi hệ thống đang kết khối từ một luồng và lập lịch luồng để thực hiện trong một QT. Trong thi hành luồng trong không gian người dùng (chương trình người dùng), một QT được ấn định chia xẻ thời gian bộ xử lí như thường được làm trong bất cứ HĐH nào. Khoảng thời gian được ấn định này là đa thành phần giữa các luồng đang tồn tại. Các luồng chạy dựa trên thư viện hỗ trợ thời gian chạy luồng. Trách nhiệm của một thủ tục thời gian chạy luồng là thực hiện việc chuyển ngữ cảnh từ luồng này sang luồng khác. Mỗi lời gọi hệ thống kết khối từ một luồng đang thực hiện là không bị HĐH bẫy lỗi nhưng được gửi tới một thủ tục thời gian chạy. Thủ tục thời gian chạy sẽ đơn giản khi giữ lại TCB của luồng gọi và tải (nạp) TCB của luồng mà nó lựa chọn tạo thành các thanh ghi phần cứng (bộ đếm chương trình, các thanh ghi và các con trỏ stack) với giả thiết rằng nó được phép thực hiện như các thao tác đặc cách. Và kết quả, không một kết khối thực sự trong hệ thống xuất hiện nhưng một luồng bị kết khối trong hàng đợi được duy trì bằng thư viện hỗ trợ thời gian chạy, và sự thực hiện QT lại được tiếp tục với một luồng khác.

Việc chuyển ngữ cảnh luồng yêu cầu một tải rất nhỏ vì nó bao hàm việc lưu giữ và khôi phục chỉ bộ đếm chương trình, các con trỏ stack. Hơn nữa, việc lập lịch chạy luồng được thực hiện bằng Thư viện thời gian chạy, người dùng có quyền lựa chọn mức ưu tiên tới luồng được tạo. Lập lịch cho luồng thông thường là theo không ưu tiên và dựa theo quyền ưu tiên vào trước thì phục vụ trước (*FCFS - First Come First Served*); Nó có thể là lập lịch có ưu tiên theo các mức khác nhau khi luồng mới được tạo có mức ưu tiên cao hơn. Sơ đồ có ưu tiên, chẳng hạn việc thực hiện cuộn (RR: Round Robin) các luồng sẽ khó hơn khi không sử dụng ngắt đồng hồ và thật sự là không cần thiết ở mỗi mức luồng. Nếu cần, một luồng có thể bao gồm nguyên thuỷ luồng *ngủ* hoặc *nhường* cho phép từ bỏ sự thực hiện của một luồng tới luồng khác nhằm tạo ra tính không đồng bộ chạy luồng. Các nguyên thủy luồng có trong các gói luồng điển hình là:

- Quản lí luồng để thực hiện việc tạo luồng, tạm dừng, kết thúc luồng.
- Ấn định ưu tiên và các thuộc tính luồng khác.
- Hỗ trợ đồng bộ và truyền thông chẳng hạn như semaphore, monitor, và CTĐ.

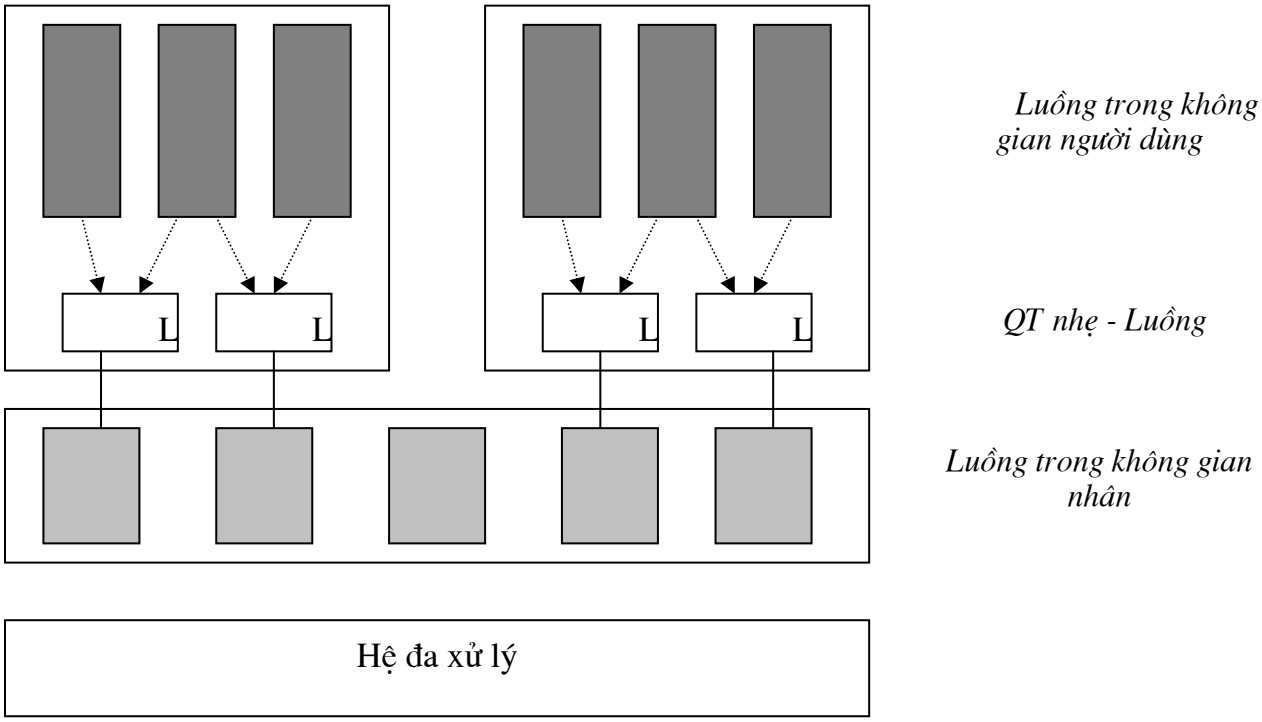
3.1.3 Thi hành luồng trong không gian nhân của hệ điều hành

Các gói luồng được thi hành như một mức phần mềm trong không gian người dùng là dễ thực hiện và cơ động mà không đòi hỏi phải thay đổi nhân. Luồng có thể được thi hành ở mức nhân với một số mở rộng. Khi thi hành luồng trong không gian nhân, việc kết khối và lập lịch luồng được xử lí như thông thường nhưng lại mềm dẻo hơn và hiệu quả hơn. Ví dụ, luồng có thể được ưu tiên một cách dễ dàng, một luồng phát ra một lời gọi hệ thống thì nó có thể bị kết khối mà không kết khối các luồng khác thuộc cùng QT và mỗi luồng có thể hoàn thành một chu trình của bộ xử lí với cùng cơ sở của các QT. Tuy nhiên, sự trừu tượng hai mức tinh vi đối với đồng thời trở nên mờ nhạt hơn và lợi thế tải chuyển ngữ cảnh luồng của QT nhẹ không còn nữa. Tính cơ động và hai mức trừu tượng đồng thời đôi khi nảy sinh thêm bất lợi khác.

Giống như các khái niệm mô hình Client/Server và RPC, luồng là khái niệm thiết kế hệ thống cơ bản. Việc lựa chọn thi hành luồng trong không gian nhân hay trong không gian người dùng là một nhân tố thiết kế hệ thống khó tính. Cách kết hợp thi hành luồng cả không gian người dùng và không gian nhân như trong Sun's Solaris hội tụ được các

lợi điểm của hai hướng tiếp cận trên. Phần dưới đây minh họa việc kết hợp hỗ trợ nhân đa luồng như vậy.

Nhân truyền thống là những luồng đơn. Thường thì chỉ có một bộ xử lí và cấu trúc của nhận là rất gọn. Vì vậy, đòi hỏi dịch vụ nhân chạy trong một luồng đơn không cần tính sự ưu tiên. Không đồng bộ là điều cần thiết trong thao tác nhân. Gần đây, hai khuynh hướng quan trọng đã được nảy sinh cho những hệ thống hiện đại. Thứ nhất, máy tính đơn với nhiều bộ xử lý đã trở nên thông dụng. Thứ hai là sự phức tạp về nhu cầu phần mềm đòi hỏi phải tạo ra nhiều dịch vụ mới trong nhân. Nhân đa luồng hỗ trợ cho những dịch vụ nhân đồng thời đã được khẳng định.



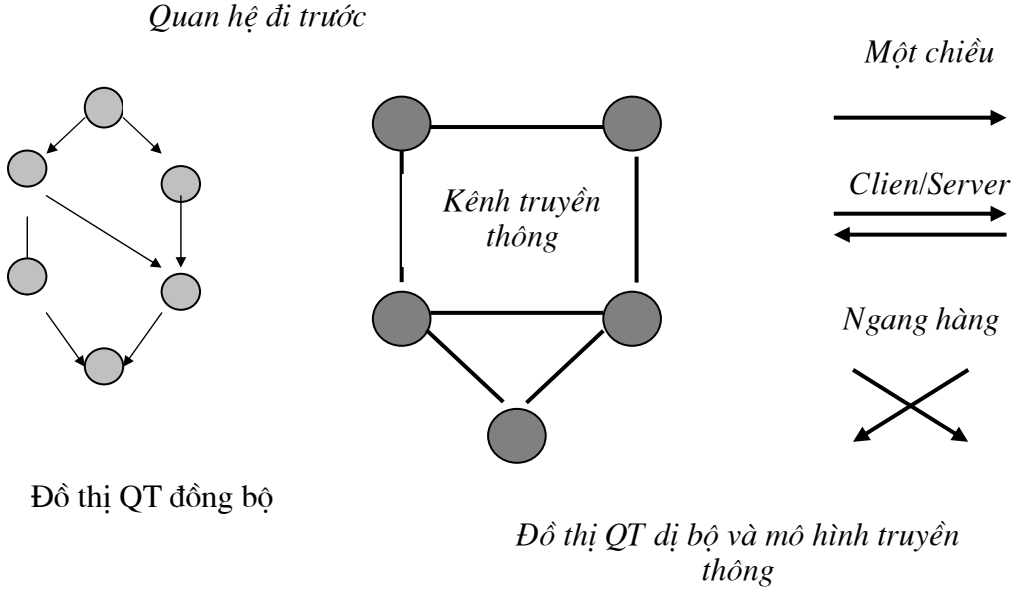
Hình 3.5. Tính đồng thời ba mức của nhân đa luồng có ưu tiên

Thao tác nội tại của nhân và những dịch vụ mà nhân cung cấp tới ứng dụng của người dùng có thể được thi hành như luồng. Luồng trong không gian nhân đã được phức hợp trong một hệ đa bộ xử lý hạ tầng. Việc thực hiện luồng là song song thực sự và có thể định ưu tiên. Sự đồng bộ giữa những luồng trong nhân trở nên cần thiết và có thể thực hiện được bằng cách dùng bộ nhớ chia xẻ. Để kết hợp chặt chẽ luồng trong không gian người dùng và luồng trong không gian nhân trong cùng một hệ thống, Solaris giới thiệu một khái niệm luồng mức trung gian và gọi là QT nhẹ LWP. LWP được QT người sử dụng tạo ra và được chương trình con (trong thư viện) thời gian chạy luồng quản lý. Chúng được nhân tổ chức như là đơn vị cơ sở cho việc lập lịch. Luồng người dùng, không thể được nhân nhận biết, cũng được tạo ra và quản lý bởi bó luồng. LWP phục vụ giao diện luồng người dùng và luồng nhân. Hình 3.5 thể hiện nhân đa luồng có ưu tiên với ba mức đồng thời: Luồng của người sử dụng là đa thành phần theo LWP trong cùng một QT; LWP là đa thành phần theo luồng nhân và luồng nhân là đa thành phần trong hệ đa bộ xử lý.

Luồng người dùng có thể được lập lịch tới bất cứ một LWP nào được QT tạo ra. Khi được gắn tới một LWP, nó trở thành thực hiện được khi dùng thời gian được nhân đã định vị tới LWP. Mỗi LWP lại được kết nối tới một luồng nhân. Lời gọi kết khối từ một luồng người sử dụng sẽ bẫy tới một LWP. LWP đó tạo ra một hệ thống thực gọi đến nhân và trở thành kết khối. Việc kết khối LWP không làm kết khối toàn bộ QT do

các luồng đợi có thể được lập lịch tới LWP khác trong cùng QT. Luồng người dùng có thể được ưu tiên vì rằng các LWP có thể được ưu tiên bởi nhân. Việc thi hành lai có tính mềm dẻo và hiệu quả từ cả luồng người dùng và luồng nhân.

Rất nhiều HĐH hiện thời có hỗ trợ luồng và có một số lượng lớn các phần mềm sử dụng luồng. Sự hỗ trợ luồng trở thành một bộ phận trong HĐH ngày nay.



Hình 3.6. Mô hình đồ thị cho sự tác động giữa các QT

3.2 Mô hình đồ thị thể hiện các QT

Mô hình đồ thị

Đoạn 3.1 mô tả các khái niệm QT và luồng. Trong đoạn này, chúng ta quan tâm đến việc làm thế nào để chúng có thể đặt cùng với nhau. Các QT có quan hệ với nhau bởi điều đó cần thiết cho:

- *tính đồng bộ*: việc chạy một vài QT phải được tiếp nối theo một trình tự nào đó. Một ví dụ về sự đồng bộ là mối quan hệ đi trước (tiền tố: predecence) giữa hai QT. Mối quan hệ đi trước giữa hai QT quy định rằng một QT chỉ được thực hiện khi mà những QT "đi trước" nó đã được thực hiện.

- hoặc/và *vấn đề truyền thông*: Trong nhiều trường hợp thì mối quan hệ đi trước hoặc thứ tự của các QT là không nhất thiết đối với việc các QT CTĐ cho nhau.

Hình 3.6 sử dụng mô hình đồ thị để biểu thị hai cái nhìn khác nhau về sự tác động giữa các QT. Đồ thị QT đồng bộ theo mô hình đồ thị có hướng không chu trình thể hiện trực tiếp mối quan hệ đi trước và thứ tự trong tập các QT. Những cạnh vô hướng trong đồ thị QT dị bộ thể hiện đường truyền thông và sự phụ thuộc giữa các QT.

Các đồ thị trong hình 3.6 chưa cho biết cụ thể thực sự mô phỏng tác động trong một hệ phức tạp. Tuy nhiên, chúng vẫn được sử dụng để xây dựng mô hình các QT và bộ xử lý trong hệ thống phân tán:

- Đồ thị QT đồng bộ được sử dụng để đánh giá tổng thời gian của một tập hợp các QT,

- Đồ thị truyền thông dị bộ có thể được sử dụng để nghiên cứu phân bố các bộ xử lí nhằm tối ưu hoá tổng chi phí về thời gian truyền thông giữa bộ xử lí.

Đồ thị quá chi tiết đối với hệ phân tán thường làm cho việc phân tích khó khăn hơn và thậm chí gần nhưng không thể giải quyết được.

Trong hình 3.6, cạnh có hướng trong đồ thị đi trước được giải thích qua truyền thông đồng bộ đối với QT gửi và nhận TĐ. Kết quả từ QT này được chuyển đến QT liền sau nó như là một input. Sự chuyển thông tin xẩy ra và được đồng bộ chỉ khi hoàn thành một QT và bắt đầu một QT tiếp theo.

Truyền thông được xác định chính xác hơn trong đồ thị QT dị bộ, khi chưa thể nói về việc làm thế nào và vào lúc nào thì việc truyền thông xẩy ra, ngoại trừ việc khẳng định tồn tại đường truyền thông giữa hai QT.

QT trong đồ thị vô hướng thực hiện vô hạn định, trái lại, QT trong đồ thị có hướng thì chỉ có thể thực hiện trong một khoảng thời gian nhất định và được gọi là thời gian sống (lifetime). Có ba kiểu CTĐ cho mô hình đồ thị dị bộ: Một chiều (*one-way*), Client/Server và ngang hàng (*peer to peer*). Nếu sử dụng thuật ngữ về truyền tin thì chúng tương đương với: truyền đơn (*simple*), Bán - hai chiều (*Half - duplex*) và hai chiều (*full - duplex*):

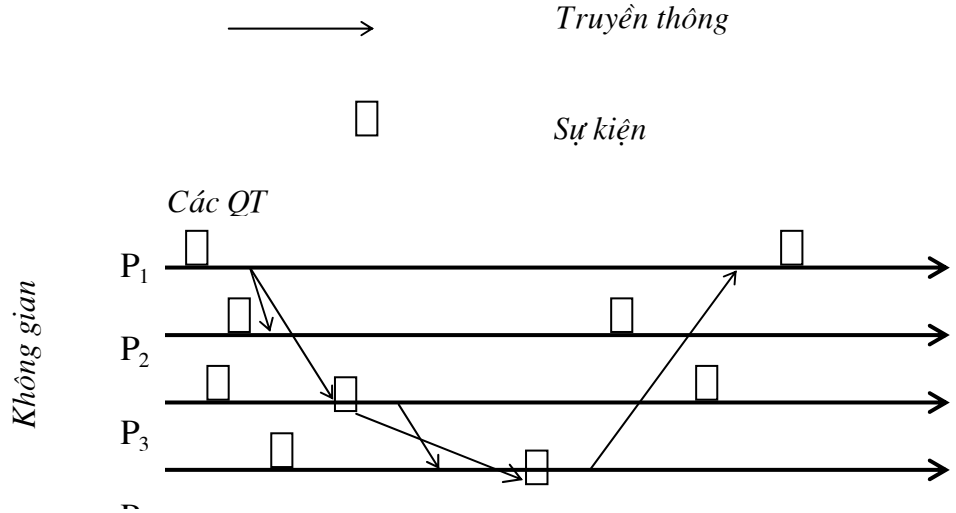
- Một QT ứng dụng CTĐ một chiều thì gửi một TĐ nhưng không có yêu cầu về sự trả lời. Ví dụ về truyền tin loại này là thông tin quảng bá (broadcast) cho mọi người mà không có sự kiểm tra từ mọi người về kết quả nhận TĐ.

- Trong mô hình Client/Server, truyền thông hai chiều: Một QT gửi yêu cầu và nhận sự trả lời đối với yêu cầu đó. Rất nhiều ứng dụng thực hiện theo quan hệ hai chiều master/slave.

- Thông tin ngang hàng là một cách trao đổi thông tin đối xứng, nó được sử dụng cho việc chuyển thông tin giữa những qúa trình cộng tác.

Mô hình không gian - thời gian

Mô hình đồ thị QT đồng bộ và dị bộ thích hợp cho việc đánh giá hệ thống nhưng lại thiếu chi tiết và đầy đủ để thể hiện được sự tương tác giữa các QT. Hình 3.7 trình bày mô hình không gian - thời gian là cách thể hiện tốt hơn việc truyền thông và quan hệ đi trước. Sự tồn tại của đường truyền thông và quan hệ đi trước giữa các sự kiện và việc truyền thông thực sự được thể hiện tường minh trong mô hình. Quan hệ đi trước hoặc là đường truyền thông tin được nhận biết dễ dàng trong mô hình không gian - thời gian. Mô hình QT này có nhiều thông tin để đánh giá sự tương tác giữa các QT hơn là thông tin để đánh giá về sự hoạt động chung toàn hệ thống.



Hình 3.7. Mô hình không gian thời gian cho sự tác động của các QT

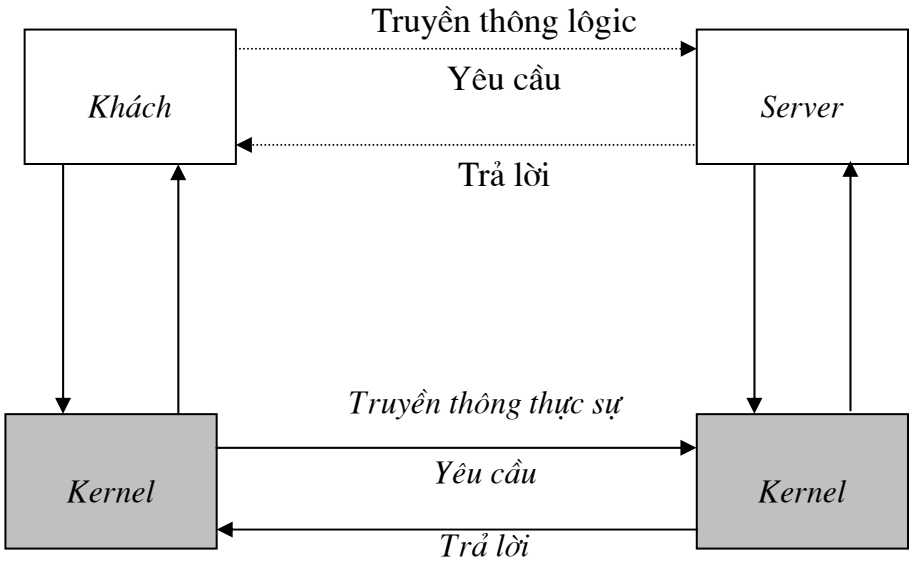
Như vậy, mỗi mô hình (đồ thị, không gian - thời gian) có tác dụng riêng và tùy thuộc vào mục đích đánh giá để chọn mô hình.

Khi các QT được thể hiện bằng đồ thị đi trước hoặc đồ thị truyền thông, sự tương tác giữa các QT phải được phát biểu trong một ngôn ngữ hoặc theo các kiểu kĩ thuật khác nhau. Giải pháp hoặc đặt ra một ngôn ngữ đồng thời (*concurrent language*) cho QT đồng thời hoặc là sẽ dễ dàng hơn khi mở rộng một ngôn ngữ tuần tự đã có bằng cách bổ sung những cấu trúc hoặc thêm một HĐH cung cấp cho việc tạo QT, truyền thông, và đồng bộ QT. Ví dụ, chúng ta đưa ra một cấu trúc điều khiển Cobegin/Coend hoặc sử dụng những lời fork/join để tạo và đồng bộ những QT đồng thời. Những QT được tạo ra bằng cách này thì được ghép chặt chẽ khi chúng có mối quan hệ chủ tớ (master/slave) hoặc là cha con (parent/child) và chia sẻ cùng một thuộc tính chung. Chúng có thể phối hợp làm việc vì một mục tiêu chung và thường được thực hiện bởi một cá thể riêng hoặc là một tổ chức nào đó.

Giải pháp đó thích hợp cho việc thực hiện mô hình đồ thị đi trước. Nhưng về lâu dài, giả sử mối quan hệ giữa các QT là ngang hàng. QT chỉ tác động cùng với QT khác thông qua truyền thông liên QT. Không có mối quan hệ đi trước giữa các QT. Trong thực tế, các QT được tạo lập một cách độc lập, chạy dị bộ và có khoảng thời gian sống khác nhau. Mô hình tốt nhất là đồ thị QT truyền thông. Trong trường hợp này, sự xác định và tương tác giữa các QT phát triển thành một HĐH thay vì thành một ngôn ngữ lớn.

3.3 Mô hình Client/Server

Một cách mô tả tác động lẫn nhau giữa các QT là mô tả theo cách các QT nhìn nhau. Mô hình phổ biến nhất là mô hình Client/Server (quan trọng gần như khái niệm trong suốt trong hệ phân tán). Mô hình Client/Server là hình mẫu lập trình thể hiện tương tác giữa các QT và cấu trúc hệ thống. Mọi QT trong hệ thống ***cung cấp những dịch vụ cho*** / hoặc ***yêu cầu dịch vụ từ*** các QT khác. QT đưa ra yêu cầu phục vụ được gọi là ***khách***, QT cung cấp dịch vụ được gọi là ***phục vụ***. Đối với mỗi tương tác, một QT chỉ có thể là khách hoặc phục vụ. Tuy nhiên, trong nhiều trường hợp, QT có thể đóng vai trò cả khách lẫn phục vụ.



Hình 3.8. Mô hình Client/Server.

Tương tác giữa khách và phục vụ thông qua dãy yêu cầu và trả lời. QT khách yêu cầu dịch vụ từ phục vụ và tự khoá bản thân lại. Phục vụ nhận được yêu cầu từ khách, thực hiện thao tác cần thiết và sau đó gửi TĐ trả lời cho khách. Khi có kết quả trả lời từ phục vụ, khách lại bắt đầu tiếp tục thực hiện. Điều cơ bản ở đây là đồng bộ hỏi - đáp để trao đổi thông tin.

Về mặt logic thì khách truyền thông trực tiếp với phục vụ nhưng thực tế thì yêu cầu hoặc trả lời phải đi qua phần nhân gửi, thông qua một mạng truyền thông đến nhân đích và QT đích. TĐ không được thông dịch bởi hệ thống. Giao thức truyền thông mức cao giữa khách và phục vụ có thể xây dựng trên những TĐ yêu cầu và TĐ trả lời. Hình 3.6 minh họa khái niệm mô hình Client/Server đối với tương tác QT.

<table>
<tr><td colspan="2">Mô hình truyền thông Client/Server</td></tr>
<tr><td>Truyền thông RPC</td><td>Truyền thông CTĐ</td></tr>
<tr><td colspan="2">Dịch vụ truyền TĐ hướng kết nối hoặc không có kết nối</td></tr>
</table>

Hình 3.9. Kiểu truyền thông Client/Server trên RPC và CTĐ

Mô hình Client/Server có thể được hiểu như một mô hình truyền thông ***hướng dịch vụ***. Đây được coi là mức trừu tượng cao của sự truyền thông liên QT, mà sự truyền thông này có thể được cung cấp (hỗ trợ) bởi hoặc là RPC hoặc truyền thông CTĐ (*message passing comminucation*) lần lượt được thi hành qua dịch vụ giao vận theo hướng kết nối hoặc không kết nối trong mạng.

Hình 3.9 cho biết quan hệ của 3 khái niệm trên đây: mô hình Client/Server, RPC và CTĐ. Những dịch vụ được cung cấp bởi phục vụ có thể theo hướng kết nối hoặc không kết nối. Một dịch vụ hướng-kết nối có thể lại được xây dựng dựa trên dịch vụ không kết nối. Nhưng điều ngược lại thì không thể. Mô hình Client/Server đã đạt được một độ trong suốt trong truyền thông.

Chương II đã giới thiệu hệ thống dịch vụ trong hệ phân tán bao gồm ba khu vực chính, đó là : Nguyên thuỷ, hệ thống và dịch vụ gia tăng giá trị.

Dịch vụ nguyên thuỷ là cơ chế nền tảng được đặt trong nhân. Từ góc độ ứng dụng thì chỉ có dịch vụ hệ thống và dịch vụ gia tăng giá trị là có thể nhìn thấy (có thể sử dụng) được từ phía người dùng.

Đối với người sử dụng thì chương trình là một tập hợp của những (QT) ***khách*** và ***phục vụ***. Nếu chúng ta thi hành dịch vụ hệ thống như là QT phục vụ và tách nó ra khỏi nhân với mọi trường hợp có thể được thì kích thước của nhân sẽ được giảm một cách đáng kể. Rõ ràng là nếu như kích thước của nhân được giảm xuống thì tính khả chuyển theo nền phần cứng khác nhau là dễ dàng hơn. Một kết quả tự nhiên là sử dụng mô hình Client/Server là QT chỉ cần một kiểu lời gọi hệ thống đến nhân đơn, chính là lời gọi gửi và nhận yêu cầu. Vì vậy, nhân không cần thiết phải phân tích cú pháp lời gọi hệ thống và xác định cái gì cần phải làm. Thay vào đó, trách nhiệm của QT phục vụ là thông dịch thông điệp theo hiểu biết nhiều nhất của nhân về cấu trúc của TĐ. Giao diện giữa QT và nhân trở nên đơn giản và đồng nhất.

Nhiều phục vụ có thể cùng tồn tại nhằm cung cấp cùng một dịch vụ. Chúng cần được định danh hoặc theo tên hoặc theo chức năng mà chúng cần thực hiện. Đòi hỏi này phục vụ việc định vị các phục vụ. Những phục vụ, được gọi là những phục vụ ràng buộc hay phục vụ đại lý, chúng ràng buộc QT khách với những QT phục vụ được chọn thành cặp, đôi khi chúng cũng cần được định vị. Cuối cùng, cần hạn chế một cách tối thiểu các phục vụ mà hoàn toàn đã biết tên hoặc địa chỉ. Khi có yêu cầu từ phía khách, phục vụ ràng buộc có thể chọn phục vụ nào thích hợp nhất cho khách đó hoặc là một phục vụ nào đó làm cân bằng tải đối với các phục vụ. Như một sự lựa chọn, cũng có thể thực hiện việc xác nhận của khách cho phục vụ.

3.4 Các dịch vụ thời gian

Mô hình không gian - thời gian tường minh tương tác giữa các QT, các sự kiện là được ghi nhận chi tiết theo đồng hồ của riêng QT đó. Trong thực tế thì đồng hồ thường được sử dụng để thể hiện thời gian (một độ đo tương đối về thời gian so với một điểm thời gian làm mốc) và bộ đếm thời gian (một độ đo tuyệt đối cho khoảng thời gian) được dùng để mô tả tính đồng thời của các sự kiện theo ba cách khác nhau:

1. Khi nào thì sự kiện xuất hiện.
2. Sự kiện xuất hiện trong bao lâu.
3. Sự kiện nào xuất hiện trước nhất.

Đối với các ứng dụng máy tính, chúng ta cần ý niệm rõ ràng về thời gian và đo thời gian. Ví dụ chúng ta cần biết một file đã được sửa đổi lần cuối cùng vào lúc nào, một khách được đặc quyền bao lâu để truy nhập phục vụ và sửa đổi nào của đối tượng dữ liệu là xẩy ra đầu tiên. Trong trường hợp thể hiện thời gian bằng đồng hồ được tăng một cách đều đặn thì không có sự nhập nhằng về sự xuất hiện các sự kiện trong một QT. Tuy nhiên, những QT tương tác trên những máy độc lập riêng rẽ có thể có nhận thức khác nhau về thời gian. Do không thể có sự nhất thể về đồng hồ toàn cục nên rất khó khăn phối hợp các hành động phân tán như thu lượm thông tin rác trên mạng, định kỳ bảo quản hệ thống file vào nửa đêm mỗi ngày hoặc việc xác nhận giá trị thời điểm kết thúc của việc nhận TĐ. Trong phần này sẽ mô tả hai khái niệm nền tảng về thời gian để xác định được thời gian trong hệ thống phân tán: Đồng hồ vật lý và đồng hồ lôgic. Đồng hồ vật lý là một xấp xỉ tốt của thời gian thực, được dùng để đo cả về thời điểm và lẫn khoảng thời gian. Đồng hồ logic được dùng để sắp xếp các sự kiện. Cả hai đều có vai trò quan trọng trong hệ phân tán.

3.4.1 Đồng hồ vật lý

Trong mọi hệ thống máy tính, đồng hồ vật lý (physical clocks) được sử dụng để đồng bộ và lập lịch cho các hoạt động của phần cứng. Mặt khác, theo khía cạnh phần mềm, nó cần thiết để mô phỏng thời gian thực hoặc là đo khoảng thời gian. Bộ đếm thời gian phần mềm dựa vào bộ đếm thời gian phần cứng. Trong hệ phân tán, mỗi đồng hồ chạy theo một nhịp riêng của mình, và vì vậy tồn tại một độ trễ trong việc trình diễn đồng hồ thời gian. Vì thông tin về thời gian không thể chuyền và nhận được một cách tức thời, do đó, một đồng hồ vật lý tuyệt đối theo lý thuyết thì không thể có. Vì vậy, chúng ta phải đặt một cái ổn định xấp xỉ thời gian thực toàn cục. Thách thức đặt ra là làm sao cho mọi máy tính có thể nhận được thời gian đồng nhất. Mong muốn có thể đạt thời gian đồng nhất gần với thời gian thực nhất có thể được. Để giải quyết vấn đề trên đây, cần một thuật toán về đồng bộ đồng hồ. Hình 3.10 thể hiện kỹ thuật dịch vụ thời gian gần giống với dịch vụ thời gian phân tán DTC (*distributed time service*) có trong DCE. Bộ ghi nhận thời gian (TC) trong mỗi máy khách yêu cầu dịch vụ thời gian tới một hoặc nhiều phục vụ thời gian (TS). Phục vụ thời gian lưu giữ những thông tin thời gian mới nhất và có thể truy cập đến nguồn thời gian thực toàn cầu. Phục vụ thời gian có thể trao đổi thông tin thời gian, vì vậy dịch vụ thời gian cua nó có thể thích hợp với những khách của nó. Tồn tại hai vấn đề cần quan tâm trong thực tế trong thi hành dịch vụ thời gian, đó là độ trễ trong việc ghi nhận thông tin về thời gian phải được bù vào và sự khác nhau giữa các nguồn thời gian phải được định cỡ.

Phần bù độ trễ

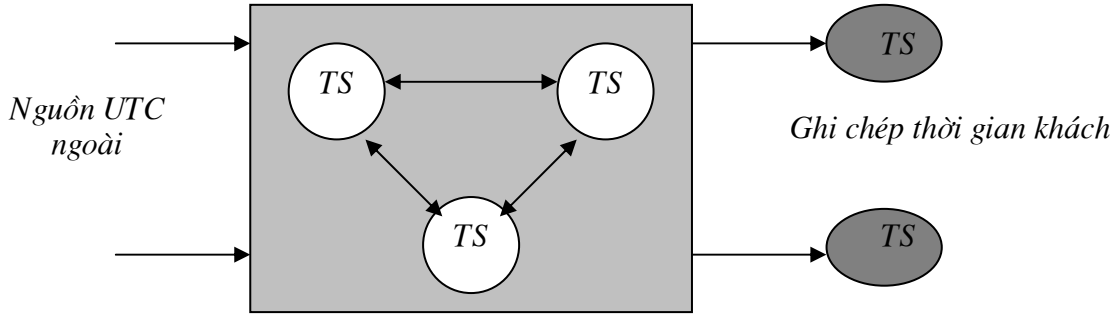
Hình 3.10 mô tả ba kiểu của truy cập thời gian: Phục vụ thời gian đến nguồn thời gian toàn cầu, khách đến phục vụ thời gian và phục vụ thời gian lẫn nhau. Nhiều nguồn hệ thống thời gian toàn cầu UTC (*Universal Coordination Time*) chuẩn có sẵn đối với

máy tính và những ứng dụng gắn chặt tới thời gian khác. Viện tiêu chuẩn và Công nghệ quốc gia NIST của Mỹ cung cấp cách truy nhập với độ chính xác lên tới một miligiây.

Dịch vụ thời gian máy tính tự động ACTS (*Automated Computer Time Service*) cung cấp những dịch vụ modem tới thời gian NIST thông qua đường điện thoại. ACTS được thiết kế cho những ứng dụng chỉ yêu cầu những *dịch vụ thời gian không thường xuyên*: QT quay số modem là quá chậm đối với việc đồng bộ những hoạt động phần cứng.

Đối với những *truy nhập mang tính thường xuyên*, NIST thực hiện một trạm phát sóng ngắn WWV thực hiện việc tán phát những tín hiệu UTC. Độ trễ thời gian của TĐ có thể được tính toán một cách chính xác nếu như khoảng cách từ trạm phát sóng và khoảng cách đến điểm truyền thông tin là được biết. Tuy nhiên, điều không may là sóng radio lại rất nhạy cảm với môi trường.

Một phương án khác là sử dụng dịch vụ của *hệ thống định vị toàn cầu GPS* (*Global Positioning System*). Tuy nhiên, vệ tinh GPS lại có quỹ đạo chậm và khoảng cách của nó đến trái đất cũng thay đổi theo thời gian. Để tính được chính xác độ trễ (hoặc khoảng cách) có thể thì cần đến sự theo dõi của nhiều vệ tinh GPS. Giá thành cho phần cứng cũng như giá thành liên quan đến việc tính toán sẽ là rất cao. Cũng có thể dựa vào trạm vệ tinh để quảng bá các thông tin UTC. Khoảng cách vệ tinh đến trạm máy tính dưới đất thì hoàn toàn cố định nhưng độ trễ tốc độ truyền thì lại rất lớn, khoảng 125 milli giây. Nhiều kênh truyền hình cáp (*Cable TV chanel*) cũng mang cả thông tin về thời gian trong tần số. Mọi nguồn thời gian toàn cầu (*Universal time source*) chứa đựng những lập luận tán thành và phê phán trong đó. Rất may là không phải tất cả các phục vụ thời gian cần truy nhập đến UTC từ những nguồn đó mà một phục vụ thời gian có thể truyền bá thời gian UTC hiện tại được nó nắm giữ đến những phục vụ thời gian



Hình 3.10. Một kiến trúc dịch vụ thời gian phân tán

khác một cách chính xác và nhanh chóng.

Vấn đề độ trễ trong QT trình diễn hoặc thu nhận thông tin UTC từ phía khách của một phục vụ thời gian lại là một vấn đề khác. Thêm vào độ trễ của QT truyền tín hiệu là độ trễ trên đường truyền thông mạng. Độ trễ trên mạng thay đổi thường xuyên và là một vấn đề đáng quan tâm hơn là độ trễ truyền tín hiệu. Giả sử T_s và T_r là thời gian gửi và nhận được những yêu cầu về dịch vụ thời gian từ khách đến phục vụ thời gian. Giả sử t_p là thời gian gian cần thiết để dịch thời gian thực hiện yêu cầu đó. UTC từ phục vụ thời gian trả về cho khách có thể được điều chỉnh cho đúng bằng cách cộng thêm một nửa của độ trễ $T_r - T_s - t_p$. Công thức tính sự bù đó dựa trên giả thiết là QT giao thông trên mạng *(network trafic*) là đối xứng.

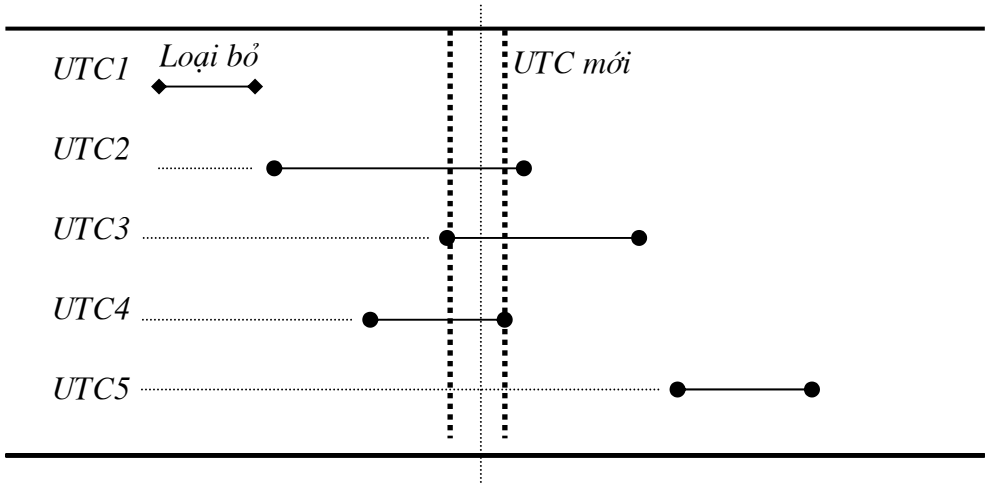
Nếu đồng hồ ở máy khách nhanh hơn UTC mới thì nó sẽ được làm chậm lại bằng phần mềm. Đồng hồ thời gian không thể quay lại được vì điều đó phủ nhận thời gian của các sự kiên trước đó. Vấn đề đồng hồ chậm hơn thì không đáng ngại nhưng tốt nhất là tăng

tốc độ đồng hồ để nó đạt được cùng với UTC một cách từ từ. Ví dụ một sự tăng đột ngột đồng hồ có thể loại bỏ QT đang đợi hoặc là nguyên nhân làm nảy sinh vấn đề hết thời gian (*time - out*).

Truy cập UTC từ một khách tới một phục vụ thời gian là một mô hình ***dịch vụ kéo*** (một kiểu dịch vụ bị động). Một phục vụ thời gian cần phải đóng vai trò chủ động trong việc TĐ UTC đến những khách của nó. Mô hình ***dịch vụ đẩy*** (dịch vụ thời gian tích cực) cho ưu điểm là duy trì được mức độ cao tính nhất quán của đồng hồ. Kiểu đẩy giống như sóng radio hoặc là TĐ vệ tinh những UTC mong đợi, cái mà mọi khách đang chờ đón trả lời. Tuy nhiên, khách lại không có cách nào để xác định được độ trễ của mạng. Điều trở ngại này làm cho giải pháp này chỉ thích hợp với những hệ thống có phần cứng đa tán phát, nơi mà độ trễ CTĐ có thể ngắn hơn và có thể dự đoán được trước. Cả hai chế độ kéo và đẩy có thể cùng áp dụng trong việc truyền thông giữa các phục vụ thời gian.

Vấn đề dự đoán độ trễ mạng cần phải đạt độ chính xác, đặc biệt khi giao thông trên mạng trở nên đông và tắc nghẽn sẽ dẫn đến kết quả trái ngược nhau về phục vụ thời gian. Để làm tăng độ nhất quán, các phục vụ thời gian có thể định cỡ UTC của chúng với những phục vụ thời gian khác. Một khách có thể nối với nhiều phục vụ thời gian để xác định tính không nhất quán của các UTC.

Xác định sự không đồng nhất



Hình 3.11. Khoảng UTC trung bình

Sự không đồng nhất giữa các phục vụ thời gian có thể được hạn chế nhờ vào sự cộng tác của các UTC. Phục vụ thời gian có thể trao đổi với các phục vụ thời gian khác theo cách kéo hoặc đẩy. Quyết định UTC dựa theo giá trị lớn nhất, nhỏ nhất, điểm giữa hoặc là trung bình của UTC. Hai thuật toán sau là lựa chọn tốt nhất cho mục đích đồng nhất hoá. Nếu trung bình được sử dụng thì hai giá trị nhỏ nhất và lớn nhất được bỏ đi (theo đúng phương pháp thống kê).

Có một điều không chắc chắn nhỏ nữa về UTC được phục vụ thời gian nắm giữ. Phục vụ thời gian có thể kết xuất một khoảng thời gian, UTC $\pm \Delta l$, trong đó Δl là một thông tin được thống kê một cách không chính xác hoặc những khoảng thời gian không chắc chắn.

Việc xác định tính không chính xác giúp khách quyết định được UTC có đáp ứng được độ chính xác cho ứng dụng đó hay không. Trung bình của UTC trong khoảng thời gian có thể được sửa lại như như trong hình 3.11. Những khoảng không kế tiếp có thể bị bỏ

đi. Những phần giao gồm nhiều UTC nhất thì được xác nhận. Điểm UTC mới được xác định ở chính giữa đoạn giao đó.

Thậm chí ngay khi đã có phục vụ thời gian nhất quán, thì tính toán UTC của khách vẫn không nhất quán do độ trễ truyền thông trên mạng không dự đoán được. Vấn đề không nhất quán của khách có thể được giải quyết nếu khách theo chiến lược như phục vụ thời gian là kết nối tới nhiều phục vụ thời gian và định cỡ UTC.

Đồng hồ vật lý đóng vai trò quan trọng trong việc phát triển phần mềm phân tán bởi vì có rất nhiều giao thức phần mềm dựa vào time-out để nắm giữ loại trừ. Nếu khởi tạo time-out bởi một QT được đặt dưới sự kiểm tra của một QT khác đặt trên một máy khác, hai đồng hồ vật lý phải đồng bộ có thể chấp nhận được đối với cả hai QT. Tem thời gian vật lý được dùng để khử thông điệp bội (ngăn ngừa phát lại) và kiểm tra sự mãn hạn quyền hạn đối với điều khiển truy nhập.

3.4.2 Đồng hồ logic

Đồng hồ vật lý là gần tương đương với đồng hồ thời gian thực toàn cục. Việc đo khoảng thời gian là hữu dụng và nhận được trực tiếp từ đồng hồ vật lý. Nói chung, có thể sử dụng đồng hồ vật lý để chỉ ra được sự kiện nào xảy ra trước sự kiện nào trừ khi chúng xảy ra rất gần nhau. Nếu như độ không chắc chắn của UTC là cao hoặc là khoảng thời gian của các sự kiện là giao nhau thì đồng hồ vật lý không cho khả năng xác định được thứ tự của các sự kiện. Đối với nhiều ứng dụng, các sự kiện không cần lập lịch hoặc đồng bộ với thời gian thực mà chỉ quan tâm đến trình tự thực hiện các sự kiện. Trong trường hợp đó thì đồng hồ lôgic được dùng để xác định thông tin về thứ tự của các sự kiện, đặc biệt trong hệ phân tán, việc duy trì một đồng hồ vật lý chung giữa các QT cộng tác là việc rất khó khăn. Đồng hồ logic Lamport là một khái niệm cơ bản để xếp thứ tự các QT và sự kiện trong hệ thống phân tán.

Mỗi một QT P_i trong hệ thống duy trì một đồng hồ logic C_i, Lamport định nghĩa ký hiệu đại số $\rightarrow$ như quan hệ xảy ra trước (*happens - before*) để đồng bộ đồng hồ logic giữa hai sự kiện. $a \rightarrow b$ có nghĩa là sự kiện a xảy ra trước sự kiện b. Trong cùng một QT, nếu sự kiện a xảy ra trước sự kiện b thì đồng hồ logic $C_i(a)$ và $C_i(b)$ được gán sao cho $C_i(a) < C_i(b)$. Đồng hồ logic trong một QT luôn được tăng một số dương tuỳ ý khi sự kiện trong QT được tăng tiến (nghĩa là thời gian không bao giờ quay trở lại và chỉ đo tương đối đối với đồng hồ logic). Một QT tương tác với một QT khác qua cặp hai phép toán gửi (*send*) và nhận (*receive*) từ quá trình P_i đến QT P_j. Việc gửi đi phải được thực hiện trước việc nhận được. Do vậy, giữa sự kiện *gửi* từ QT P_i và sự kiện *nhận* tại QT P_j phải đảm bảo tính chất là C_i (gửi) < C_j (nhận) do QT nhận không thể hoàn thành được trước khi sự kiện gửi chưa được thực hiện. Đồng hồ logic C dựa trên quan hệ xảy ra trước được tổng kết theo hai quy tắc sau:

1. Nếu $a \rightarrow b$ trong cùng một QT thì $C(a) < C(b)$.
2. Nếu a là sự kiện gửi một TĐ của QT P_i và b là sự kiện nhận cũng TĐ đó của QT P_j thì $C_i(a) < C_j(b)$.

Quy tắc 1. rất dễ dàng được thi hành vì trong cùng một QT (chẳng hạn, mỗi khi xuất hiện một sự kiện mới thì bộ đếm đồng hồ lôgic của QT đó tăng lên 1). Quy tắc 2. có thể có hiệu lực nếu như gắn tem thời gian đồng hồ logic của QT gửi vào trong TĐ và QT nhận sẽ cập nhật đồng hồ logic của mình bằng cách sử dụng thời gian của đồng hồ của chính nó và việc tăng tem thời gian theo công thức:

$$C(b) = C(a) + d \text{ và } C_j(b) = \text{Max } (TS_a + d, C_j(b))$$

trong đó TS_a là tem thời gian của sự kiện được gửi và d là một số dương. Mỗi quan hệ xảy ra trước thể hiện kết quả của hai QT có tính bắc cầu:

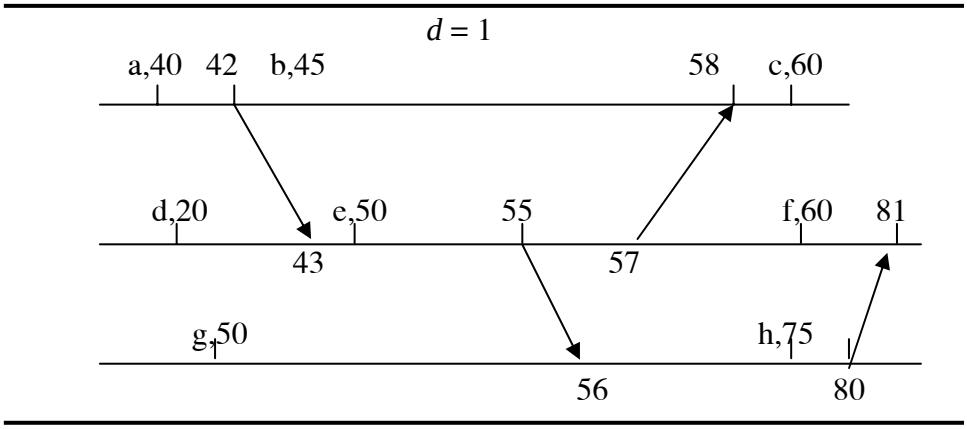
Nếu a $\rightarrow$ b và b $\rightarrow$ c thì a $\rightarrow$ c.

Hai sự kiện a và b được gọi là hai sự kiện rời nhau và có thể chạy đồng thời nếu như không cả a$\rightarrow$ b và b$\rightarrow$ c. Biểu đồ không gian thời gian trong hình 3.12 thể hiện mối quan hệ của các sự kiện {(a,e,c) và (d,e,h)} và những sự kiện đồng thời {(b,e), (f,h)}. Đồng hồ logic cho những sự kiện đồng thời không liên quan đến những sự kiện khác.

Thứ tự bộ phận và thứ tự toàn bộ của sự kiện

Sử dụng quan hệ xảy ra-trước và hai quy tắc trên, mọi sự kiến có quan hệ nhân quả sẽ được sắp xếp bởi đồng hồ logic. Kết quả này chỉ cho một thứ tự bộ phận trong đồ thị sự kiện. Với hai sự kiện rời nhau a và b (của hai QT i và QT j), thì $C_i(a) < C_j(b)$ không có nghĩa là a$\rightarrow$ b. Hơn nữa, có khả năng là $C_i(a) = C_j(b)$. Thứ tự toàn cục của các sự kiện có thể nhận được bằng cách thêm một quy tắc cho hai sự kiện rời nhau (các sự kiện nhân quả luôn có thứ tự).

3. Với mọi sự kiện a và b thì $C(a) \neq C(b)$.



Hình 3.12. Mối quan hệ xảy ra trước và sự gán thời gian đồng hồ

Những sự kiện rời rạc cùng với đồng hồ logic định danh có thể được phân biệt theo mốc đồng hồ logic của chúng với một số hiệu QT hoàn toàn khác nhau, điều đó đảm bảo sự duy nhất của một đồng hồ logic toàn cục cho cả hệ thống. Thứ tự của các sự kiện rời nhau không liên quan tới việc thực hiện chính quy của QT. Tập thứ tự toàn cục các sự kiện mô tả một dãy thực hiện đúng đắn mềm dẻo của các sự kiện. Tồn tại nhiều thuật toán điều khiển đồng thời sử dụng tính chất thứ tự toàn cục của sự kiện dựa trên đồng hồ logic.

3.4.3 Đồng hồ logic vector

Thậm chí thứ tự sự kiện toàn cục sử dụng đồng hồ logic được miêu tả ở trên không thể khẳng định được là thực sự có phải sự kiện a xảy ra trước sự kiện b hay không cho dù $C(a) < C(b)$ bởi vì chúng có thể cùng được thực hiện. Trong trường hợp đó, cần sử dụng đồng hồ logic vector, trong đó theo phương thức đồng hồ logic vector, mỗi QT lưu giữ một vertor đồng hồ logic riêng đối với mỗi sự kiện.

Giả sự đồng hồ logic vector của sự kiện a tại bộ xử lý i là $VC_i(a) = \{TS_1, TS_2, ..., C_i(a), .., TS_n\}$, trong đó n là số QT đồng thời, $C_i(a)$, còn được ghi là TS_i, là thời gian đồng hồ logic của sự kiện a trong QT P_i và TS_k (k nhận 1, 2, ... , n ngoại trừ i) là ước lượng tốt

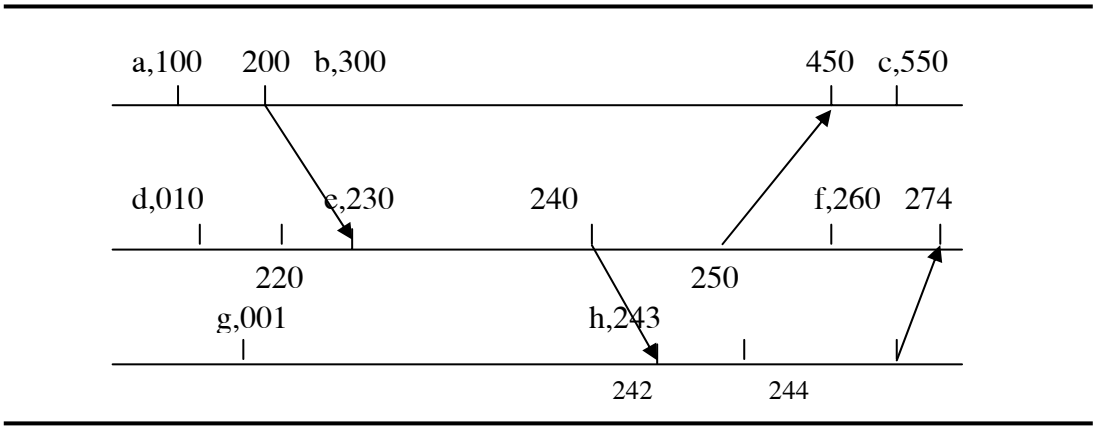
nhất cho thời gian đồng hồ logic của QT P_k. Ước lượng tốt nhất cho thời gian đồng hồ lôgic của QT khác nhận được thông qua thông tin về tem thời gian được mang trong các TĐ trong hệ thống.

Với mỗi QT thì đồng hồ logic vector được khởi tại bằng 0 tại thời điểm bắt đầu thực hiện QT:

- Đồng hồ logic trong nội tại QT được tăng như quy tắc 1.

- Quy tắc 2 được biến đổi theo cách như sau: Khi QT P_i gửi TĐ m (sự kiện a) đến QT P_j, tem thời gian logic của m (chính là $VC_i(m)$) cũng được gửi cùng với m. Giả sử b là sự kiện nhận m tại QT P_j. P_j sẽ cập nhật đồng hồ logic vector $VC_j(b)$ với $TS_k(b) = Max \{ TS_k(a), TS_k(b)\}$. P_j sẽ giữ giá trị lớn nhất trong cặp của với k = 1 ... n và tăng đồng hồ logic vector của nó lên theo kết quả tính toán. Bằng cách đó, mọi thông tin về đồng hồ được chuyền đến tất cả các QT bằng cách gửi các tem thời gian TS_i trong TĐ.

Rõ ràng rằng, nếu sự kiện a trong QT P_i xảy ra trước sự kiện b trong quá P_j thì $VC_i(a) < VC_j(b)$, nghĩa là $TS_k(a) \le TS_k(b)$ với mọi k và $TS_j(a) < TS_j(b)$. Điều đó xẩy ra do có một đường chuyển nhân quả từ sự kiện a đến sự kiện b và sự kiện b có nhiều thông tin cập nhật hơn sự kiện a, tem thời gian được truyền dọc theo đường đó và quy tắc để cập nhật luôn là chọn cái lớn hơn trong hai cái. Thêm vào nữa, đồng hồ logic của sự kiện kế tiếp sẽ được tăng bởi sự kiện a. Vì vậy, $TS_j(b)$ phải lớn hơn $TS_j(a)$. Đối với những sự kiện rời rạc thì không thể có $VC_i(a) < VC_j(b)$ trừ khi $a \rightarrow b$ bởi vì QT P_i (nơi xảy ra sự kiện a) sẽ được cập nhật một cách tốt cho thời gian của mình hơn mọi ước lượng của các QT khác về thời gian hiện tại của QT P_i. Do đó, $C_i(a)$ lớn hơn hoặc bằng với TS_i trong những vector khác. Cũng như vậy, $VC_j(b) < VC_i(a)$ nếu như $b \rightarrow a$. Nói tóm lại là chúng ta có thể kết luận là hai sự kiện có thể có hay không mối quan hệ trước sau bằng cách so sánh vector thời gian của hai sự kiện đó.



Hình 3.13. Đồng hồ logic vector

Nếu $VC_i(a) < VC_j(b)$, chúng ta có thể kết luận là sự kiện a xảy ra trước sự kiện b. Nếu không thì a và b đồng thời. Hình 3.11 đưa ra một ví dụ của đồng hồ logic vector dùng mô hình không gian thời gian.

3.4.4 Đồng hồ logic ma trận

Khái niệm đồng hồ logic vector có thể được mở rộng thành đồng hồ logic ma trận (*Matrix logical clock*). Một đồng hồ ma trận MC[k,l] tại quá trình P là một ma trận cấp nxn, nó thể hiện giờ logic bằng vector của đồng hồ logic vector. Dòng i trong ma trận MC[1..n,1..n] là một đồng hồ logic vector của P_i. Dòng thứ j trong ma trận

MC[1..n,.1..n] xác định chính tri thức mà quá trình P_j có được về đồng hồ logic vector của QT P_i. Luật cập nhập đồng hồ logic ma trận giống như cập nhật cho đồng hồ logic vector. Mỗi một sự kiện địa phương, QT sẽ tăng đồng hồ của nó bằng cách đặt

$MC_i[i,i] = MC_i[i,i] + d.$

Khi QT P_i gửi TĐ đến QT P_j, toàn bộ đồng hồ ma trận $MC_i[k,l]$ được gán tem thời gian bằng $TS_i[k,l]$ và gửi cùng với TĐ đến QT P_j. Đầu tiên, P_j cập nhật đồng hồ vector bằng luật lấy lớn hơn trong một cặp.

$MC_j[j,l] = \max \{ MC_j[j,l], TS_i[j,l] \}$ $l = 1..n$

Sau đó, P_i cập nhật toàn bộ ma trận một lần nữa bằng luật lấy phần tử lớn hơn trong một cặp

$MC_j[k,l] = \max \{ MC_j[k,l] , TS_i[k,l] \}$ $k = 1..n, l = 1..n$

Lần cập nhật đầu tiên bảo quản được thứ tự của các sự kiện. Lần cập nhật thứ hai truyền thông tin cho những QT khác qua cách chuyển các TĐ.

3.5 Cơ cấu ngôn ngữ cho đồng bộ

Trên cơ sở khái niệm QT, yêu cầu đặt ra là cần xây dựng cấu trúc ngôn ngữ thi hành được sự tương tác QT. Một ngôn ngữ lập trình đồng thời cho phép đặc tả được việc xử lý đồng thời, cách thức để đồng bộ các QT đồng thời và truyền thông giữa chúng. Theo một lẽ tự nhiên, cần xuất phát từ một ngôn ngữ tuần tự sẵn có, để rồi bổ sung thêm các phương tiện hỗ trợ xử lý đồng thời. Cách tiếp cận này là dễ dàng hơn cho người lập trình (ứng dụng) vì chỉ cần một ít bổ sung khi học ngôn ngữ. Từ một ngôn ngữ tuần tự cần phải bổ sung các cấu trúc sau đây để có thể nhận được một ngôn ngữ đồng thời:

- Đặc tả được các hoạt động đồng thời
- Đồng bộ hoá các QT
- Truyền thông liên QT
- Sự thực hiện không định trước của các QT

Đồng bộ hóa QT đã được khảo sát kỹ trong HĐH tập trung (sinh sự kiện ***generate*** / chờ đợi sự kiện ***wait***). Việc truyền thông liên QT thông qua việc CTĐ là một vấn đề mới khi lưu ý đến hệ thống phân tán. Mục này đưa ra một số giải pháp chuẩn đồng bộ QT cùng với việc làm phù hợp chúng đối với hệ phân tán và cách thức tiến hóa chúng thành vấn đề truyền thông nút trong hệ phân tán. Rất nhiều cách đặt vấn đề được đặt ra để giải quyết bài toán đồng bộ theo nhiều góc độ khác nhau của một ngôn ngữ lập trình. Đầu tiên mô tả ngắn gọn cấu trúc ngôn ngữ để từ đó tìm cách mở rộng chúng nhằm đồng bộ QT.

3.5.1 Cấu trúc ngôn ngữ

Một ngôn ngữ hướng thủ tục chung được định nghĩa tổng quát bằng việc đặc tả hoàn chỉnh cấu trúc cú pháp và ngữ nghĩa các thành phần chính. Theo đặc tính, các thành phần này được phân lớp như sau:

- *Cấu trúc chương trình* chỉ ra chương trình và các thành phần con của nó (thủ tục, khối, câu lệnh, biểu thức, biến, hằng....) được bố trí như thế nào. Ngầm định các thành phần của chương trình được thực hiện tuần tự; ngoại trừ việc thay đổi tường minh bằng câu lệnh điều khiển.

- *Cấu trúc dữ liệu* được định nghĩa để trình bày các đối tượng trong chương trình. Tính trừu tượng hoá của kiểu dữ liệu và sự thi hành hiệu quả của chúng là mục tiêu nguyên thủy.
- *Cấu trúc điều khiển* qui định dòng thực hiện chương trình. Đa số ngôn ngữ nhấn mạnh việc dùng cấu trúc điều khiển hiển dạng one - in - one - out (một - vào - một - ra: là một đặc trưng của lập trình có cấu trúc) chẳng hạn như là if - then - else, while - do, repeat - until. Một loại cấu trúc điều khiển bao chứa lời gọi, quay về và thoát khỏi chương trình con.
- *Các thủ tục và lời gọi hệ thống* kích hoạt các thủ tục đặc biệt hoặc dịch vụ hệ thống. Chúng làm thay đổi hướng thực hiện và cho phép truyền tham số.
- *Vào/Ra* cho phép nhập dữ liệu vào và đưa ra kết quả thực hiện chương trình. Mọi chương trình đều có ít nhất một thao tác ra. Vào/Ra có thể được xem là trường hợp riêng của truyền thông CTĐ.
- Phép gán sinh kết quả cho đối tượng dữ liệu: đó là các thao tác cơ bản khi thực hiện chương trình

Bảng 3.1 cho ví dụ về các phương pháp đồng bộ được hiển thị theo phương tiện sử dụng ngôn ngữ. Mối quan hệ giữa phương pháp đồng bộ và những phương tiện ngôn ngữ tương ứng là không tường minh. Nó được dùng chỉ để chứng tỏ sự tiến hóa việc phát triển cấu trúc ngôn ngữ cho ĐBQT.

Phương pháp đồng bộ	Phương tiện ngôn ngữ
Đồng bộ chia xẻ biến chia xẻ	
Semaphore (Cờ tín hiệu)	Biến chia xẻ và lời gọi hệ thống
Bộ giám sát	Trừu tượng hóa kiểu dữ liệu
Khoảng tới hạn điều kiện	Cấu trúc điều khiển
Kết xuất định kỳ	Kiểu dữ liệu và cấu trúc điều khiển
Biểu thức đường đi	Kiểu dữ liệu và cấu trúc chương trình
Đồng bộ CTĐ	
Các QT tuần tự truyền thông	Vào và Ra
Lời gọi thủ tục từ xa - RPC	Lời gọi thủ tục
Cuộc hẹn	Lời gọi thủ tục và truyền thông

Bảng 3.1 Kỹ thuật đồng bộ và phương tiện ngôn ngữ

Khái niệm đồng bộ ở đây được chia làm hai loại: 5 trường hợp trên là phương pháp đồng bộ chia xẻ biến chung, còn 3 trường hợp dưới theo cách tiệm cận CTĐ. Hai đoạn tiếp theo sẽ thảo luận về các cơ chế đồng bộ kiểu CTĐ thông qua giải bài toán ***đọc đồng thời / ghi độc quyền*** bằng cách sử dụng từng phương pháp ở đây.

Bài toán QT đọc / QT ghi sử dụng *giả thiết thông thường* về đọc và ghi thực thể đối tượng dữ liệu (chẳng hạn, nhiều QT đọc có thể đồng thời nhưng QT ghi cần loại trừ ràng buộc với các QT đọc và ghi khác: nó không cho phép QT đọc và ghi khác đồng thời thực hiện với nó). Bài toán này là đủ tổng quát đối với mô hình động bộ hóa và đồng thời trong nhiều ứng dụng.

Bài toán QT đọc / QT ghi rất đa dạng, vì thế không thể chỉ đặt chúng ở mức độ khái niệm lập trình đồng thời mà trong nhiều chuyên mục và dự án lập trình cần xác định

biến thể của chúng một cách chính xác hơn. Phương án ***ưu tiên QT đọc***: *một QT ghi xuất hiện sẽ đợi cho đến khi không còn QT đọc chạy*. Như vậy QT đọc có quyền ưu tiên cao hơn QT ghi và điều này có thể dẫn tới việc QT ghi "bị xếp xó" nếu QT đọc mới lại xuất hiện trước khi một QT đọc chưa thực hiện xong. Phương án ***ưu tiên QT ghi***: *một QT đọc xuất hiện sẽ đợi cho đến khi không còn QT ghi chạy và QT ghi đang đợi*. Điều này cũng dẫn tới tình huống QT đọc đợi mãi nhưng chẳng bao giờ đến lượt mình.

Tồn tại điểm chưa rõ ràng khi định nghĩa ưu tiên QT đọc *khi cả QT đọc và QT ghi đang đợi một QT ghi khác* đang thực hiện. Sau khi QT ghi này hoàn thành xong thì sẽ trao quyền điều khiển cho ai ? (QT đọc hay QT ghi ?). Ta gọi sự ưu tiên QT đọc là sự ***ưu tiên QT đọc mạnh*** *(strong reader)* nếu như QT đọc luôn được xếp lịch ưu tiên hơn các QT ghi đang đợi một QT ghi hoàn thành. Nếu lịch là không tường minh (không đảm bảo cái gì được xếp lịch tiếp theo) thì đó được gọi là ***ưu tiên QT đọc yếu*** *(weak reader)*. Trong trường hợp còn lại sau khi hoàn thành quyền điều khiển luôn được trao lại cho QT ghi thì được gọi là ***ưu tiên QT đọc yếu hơn*** *(weaker reader)*. Không tồn tại tính mập mờ đối với định nghĩa sự ưu tiên QT ghi vì QT đọc luôn phải đợi cho đến khi không còn QT ghi đang chạy và QT ghi nào đợi nữa.

Các lập luận dưới đây luôn giả thiết chọn *phương án ưu tiên QT đọc yếu* (tức là QT ghi phải đợi cho đến khi không còn một QT đọc hay ghi nào đang xảy ra) làm cơ sở đưa ra những ví dụ về giải pháp đồng bộ.

3.5.2 Đồng bộ hoá biến chung

Cơ chế đồng bộ biến chung đã được phát triển để đồng bộ QT đồng thời ở HĐH tập trung. Đúng như tên gọi, sự cộng tác QT được thực hiện bằng cách chia xẻ biến. TTLQT đã không đề cập tới cả CTĐ lẫn truyền thông Client/Server. Mặc dù vậy tiếp cận tập trung vẫn được sử dụng để thiết kế HĐH phân tán. Ví dụ, các luồng và bài toán (task) sử dụng bộ nhớ chia xẻ phân tán tiếp tục dùng đồng bộ bộ nhớ chia xẻ. Cũng vậy, vẫn sử dụng mô phỏng đồng bộ biến chia xẻ trong việc CTĐ.

Bảng 3.1 tóm tắt các cơ chế đồng bộ bộ nhớ chia xẻ. Năng lực và sự tương thích với phương tiện ngôn ngữ về cơ chế đã được so sánh và sự thích hợp với hệ thống phân tán được xác nhận trong bảng này.

Semaphore: Tiếp cận biến chia xẻ và lời gọi hệ thống

Semaphore là kiểu dữ liệu cài đặt trong hệ thống. Biến thuộc kiểu semaphore được gắn với một khoá và một dòng xếp hàng các QT kết khối theo mục đích chia xẻ biến chia xẻ. Chỉ với 2 toán tử P đóng khóa và V mở khóa, semaphore được dùng để ĐBQT. Hai toán tử đó được thi hành như lời gọi hệ thống tới HĐH. HĐH bảo đảm tính không thể chia tách của mỗi toán tử và chịu trách nhiệm đối với việc kết khối và tách khối QT trong dòng xếp hàng. Đặc trưng khóa của semephore là nó cung cấp cơ chế khóa nguyên thủy nhất. Sự cộng tác các QT hoạt động đạt được sự đồng bộ đúng đắn hoàn toàn thuộc về QT người dùng. Tồn tại sự không trong suốt khi ngầm định khái niệm bộ nhớ chia xẻ.

Giải pháp semaphore ở hình 3.14 cho thấy sự phụ thuộc mạnh giữa QT đọc và QT ghi. QT người dùng biết được sự tồn tại của các QT khác và đây là giả thiết không mong muốn (vì sẽ gây rắc rối) trong hệ thống phân tán. Biến chia xẻ ***rc*** là biến bộ đếm số QT đọc còn biến semaphore ***mutex*** cung cấp sự loại trừ ràng buộc cho việc cập nhật ***rc***.

Việc mở rộng kiểu dữ liệu semaphore hệ thống thành kiểu dữ liệu semaphore người dùng tổng quát hơn được phát triển thành khái niệm kiểu giám sát ***monitor*** ở mức độ trừu tượng cao hơn.

```
Var mutex, db: semaphore; rc: integer; {rc : read counter}
Reader processes                    Writer processes
P(mutex);
rc := rc + 1;
if rc = 1 then P(db);               P(db);
V(mutex);
Read database                       Write database
P(mutex);
rc := rc - 1;
if rc = 0 then V(db);               V(db);
V(mutex);
```

Hình 3.14. Giải pháp semaphore cho bài toán ưu tiên QT đọc yếu

Khoảng tới hạn điều kiện

Khoảng tới hạn điều kiện (Conditional Critical Region *CCR*) là phương án cấu trúc điều khiển theo cách tiếp cận semaphore. Cú pháp của CCR có dạng region - begin - end. Một QT vào khoảng tới hạn, điều kiện của nó phải được kiểm tra: Nếu điều kiện đó chưa thoả mãn thì nó dừng lại và một QT khác được tiếp tục.

Hình 3.15 mô tả giải pháp CCR cho phương án ưu tiên QT đọc *yếu*. Các tiếp cận cấu trúc điều khiển *giả thiết* biến chia xẻ và *yêu cầu* biên dịch khoảng tới hạn thành *nguyên thủy đồng bộ* có sẵn trong *HĐH*. Đòi hỏi cần có không gian địa chỉ chung và việc biên dịch riêng rẽ làm cho CCR có vẻ không thích hợp đối với hệ phân tán.

```
Var db: shared; rc: integer;
Reader processes                          Writeter processes
    Region db begin rc := rc + 1; end;        Region db when rc = 0
    Read database                             begin write database end;
    Region db begin rc := rc - 1; end;
```

Hình 3.15 *Giải pháp CCR cho bài toán ưu tiên QT đọc* yếu

Monitor: ***Tiếp cận kiểu dữ liệu trừu tượng***

Monitor là một khái niệm mô hình đối tượng và có cấu trúc cú pháp giống như kiểu dữ liệu người dùng định nghĩa. Monitor gồm một khai báo các biến cục bộ của nó, một tập các thao tác được phép (hoặc thủ tục monitor) và một thủ tục khởi tạo thực hiện khởi tạo trạng thái của monitor.

Sự khác nhau giữa monitor và kiểu dữ liệu thông thường do người dùng định nghĩa chỉ là ***giả thiết ngữ nghĩa rằng chỉ một thể hiện cho một đối tượng monitor có thể hoạt động tại một thời điểm***. Giả thiết hoàn toàn tuyệt đối này hoạt động giống như sự loại trừ nhau của cặp thao tác P và V trong kiểu dữ liệu semaphore. Tuy nhiên, vùng nguy hiểm là cố định và được xác định sẵn trong các thủ tục monitor. Kết quả là, monitor có tính cấu trúc hơn semaphore. Để hoạt động cộng tác, mỗi khi QT ở trong một thủ tục monitor, *các biến điều kiện* với hai toán tử chuẩn ***wait*** và ***signal*** được dùng để cho phép tạm dừng hoặc tiếp tục thực hiện QT trong monitor. Do việc xen kẽ các thủ tục

monitor là cho phép, việc thi hành monitor cần đảm bảo rằng không có quá một QT trong monitor được hoạt động tại một thời điểm. Vì các thủ tục monitor chấp nhận tham số, monitor cung cấp truyền thông QT tốt giống như ĐBQT. Rõ ràng là, sự che khuất và chi tiết ĐBQT khỏi QT người dùng là một bước chuyển biến chính để monitor cung cấp tính năng trong suốt.

Hình 3.16 mô tả giải pháp monitor cho bài toán ưu tiên QT đọc *yếu*. Các monitor phục vụ tựa như người điều khiển quản lý các luật đối với QT đọc và ghi đồng thời. Không còn sự tương tác trực tiếp giữa QT đọc và QT ghi. Monitor tương đồng với một phục vụ, còn QT đọc và QT ghi giống như các *khách*. Đáng tiếc, điểm hạn chế chính là thậm chí nếu monitor được thi hành bởi hệ thống và chịu trách nhiệm điều khiển đọc/ghi, các hoạt động *đọc và ghi thực sư vẫn được diễn ra trong QT người dùng*. Trong trường hợp này, monitor không là một phục vụ hệ thống đầy đủ. Thêm nữa, nhu cầu của QT người dùng bắt đầu/kết thúc mỗi thao tác đọc/ghi không thể là trong suốt như thao tác đọc/ghi đơn giản.

Ngoại trừ giả thiết về chia xẻ bộ nhớ, khái niệm monitor chưa thể thích nghi đối với hệ thống phân tán. Để cung cấp tính trong suốt và đọc/ghi đồng thời cơ sở dữ liệu, hoạt động đọc/ghi thực sự phải được xảy ra trong monitor. Tuy nhiên, sự thực hiện phức của thủ tục monitor là không cho phép và chính vì thế, giải pháp monitor đa luồng có vẻ hợp lý. Mỗi luồng tương ứng với mỗi hoạt động của một thủ tục monitor mà không cần kết khối monitor. Luồng có thể tạm dừng và đợi cho tới khi có điều kiện và chúng chia xẻ các biến bằng cách dùng semaphore. Đáng tiếc, giải pháp này làm mất đi đặc trưng thống nhất đơn của thủ tục monitor, hoạt động đơn của một thủ tục monitor để loại trừ ràng buộc. Điều đó không được tiếp diễn trong monitor. Giải pháp đối với một thủ tục monitor là đôi lúc thì loại trừ và đôi lúc thì đồng thời.

```
Rw: monitor
Var rc: integer; busy: boolean; toread, towrite: condition;
procedure startread                    procedure endread
Begin                                  Begin
If busy then toread.wait;
rc := rc + 1;                          rc := rc - 1;
toread.signal;                         if rc = 0 then towrite.signal;
end;                                   end;
 procedure startwrite                  procedure endwrite
Begin                                  begin
If busy or rc # 0 then toread.wait;    busy := false;
busy := true                           toread.signal or towrite.signal;
end;                                   end;
begin      rc := 0; busy:= false;      end;
```

```
Reader process                 Writer process
Rw.strartread;                 Rw.start.write;
Read database;                 Write database;
```

```
Rw.endread;                    Rw.endwrite;
```

Hình 3.16. Giải pháp monitor bài toán ưu tiên QT đọc yếu

Serialize (Bộ giám sát): ***Tiếp cận tổ hợp trừu tượng dữ liệu và cấu trúc điều khiển***

Giải pháp đồng bộ monitor cho thấy cần phải có tính đồng thời trong monitor và trong thời gian đó duy trì tính nguyên tử của thao tác của thủ tục monitor. Serialize là mở rộng khái niệm monitor cho phép thực hiện được các điều trên. Serialize có cấu trúc tương tự như monitor và QT sử dụng lời gọi thủ tục serialize cũng giống như cách sử dụng monitor. Giống như monitor, truy nhập loại trừ là được thừa nhận. Tuy nhiên một thủ tục serialize bao gồm hai kiểu khoảng: một đòi hỏi loại trừ ràng buộc và một cho phép QT đồng thời hoạt động, loại thứ hai được gọi là khoảng rỗng (hollow). Hơn thế, serialize còn có cấu trúc điều khiển mới là ***joincrowd -then - begin - end***. Khi một QT đi vào khoảng rỗng, nó giải phóng serialize và ghép nối các QT đồng thời. Việc kết khối QT bởi ***wait*** theo biến điều kiện trong monitor được thay thế bằng thủ tục ***endqueue*** trong serialize. Việc chuyển dịch các QT trong hàng đợi khi điều kiện mong đợi của nó biến đổi được làm hoàn toàn bằng hệ thống hơn là sử dụng signal trong monitor. Dùng hàng đợi đã làm tăng thêm tính rõ ràng của chương trình. Hình 3.17. mô tả giải pháp serialize so sánh với các ví dụ trước. Serialize cho phép loại trừ ràng buộc và thực hiện đồng thời trong các thủ tục serialize. Serialize tóm gọn tốt nhất đối tượng đồng thời và hầu như tương đồng với phục vụ tài nguyên. Khách yêu cầu truy nhập cơ sở dữ liệu dùng các thủ tục serialize đọc và ghi trực tiếp và trong suốt.

```
Rw: serializer;
Var readq, writeq: queue; rcrowd, wcrowd: crowd;
Procedure read
Begin
        Enqueue(readq) until empty(wcrowd);
        Joincrowd(rcrowd) then begin read database end;
End;
Procedure write
Begin
        Enqueue(writeq) until (empty(wcrowd) and empty(rcrowd));
        Joincrowd(wcrowd) then begin read database end;
End;
```

Hình 3.17. *Giải pháp serializer cho bài toán ưu tiên QT đọc* yếu

Path Expression: ***Môt tiêm cân trừu tượng dữ liêu và cấu trúc chương trình***

Khái niệm path expression (biểu thức đường dẫn) khác hẳn so với những phương pháp đồng bộ được thảo luận ở trên. Giống như monitor, đó là kiểu dữ liệu trừu tượng. Tuy nhiên các thủ tục xác định trong kiểu dữ liệu trừu tượng path expression không chỉ tường minh tới bất kì nguyên thủy đồng bộ nào. Chỉ có thứ tự thực hiện các thủ tục phải đi sau tập ràng buộc của path expression. Path Expression là đặc tả ngôn ngữ bậc

cao mô tả các thao tác được định nghĩa như thế nào đối với đối tượng chia xẻ để có thể được gọi để đảm bảo yêu cầu đồng bộ. Vì lí do này chúng được coi như là cấu trúc chương trình do nó giống như mô tả hình thức một chương trình. Giải pháp path expression cho vấn đề ưu tiên QT đọc yếu là rất ngắn gọn :

Path 1 : ([read], write) end

Hằng số 1 ràng buộc số lượng hoạt động đồng thời trong ngoặc đơn là 1 và điều đó đặc tả sự loại trừ giữa các QT đọc và ghi. Dấu ngoặc vuông chỉ ra QT đọc có thể xảy ra đồng thời. Chương trình dịch phải sẵn sàng chuyển path expression thành dịch vụ nguyên thủy đồng bộ thi hành được. Rất nhiều mở rộng của khái niệm path expression đã được phát triển nhằm làm tăng khả năng đặc tả yêu cầu về đồng thời và đồng bộ. Ví dụ đáng kể nhất đã được khẳng định trong path expression đối với phối hợp có điều kiện.

3.5.3 Đồng bộ hóa chuyển thông điệp

Không có bộ nhớ chia xẻ nên trong hệ phân tán, CTĐ là cách truyền thôngđược lựa chọn. Đó cũng là một dạng đồng bộ *ngầm* do *TĐ có thể nhận chỉ sau khi chúng được gửi đi*. Đối với nhiều ứng dụng, nhận là kết khối còn gửi có thể kết khối hoặc không. Ta gọi gửi không kết khối-nhận kết khối là CTĐ dị bộ và gửi kết khối-nhận kết khối là CTĐ đồng bộ. Mục này trình bày cách đồng bộ khi sử dụng cho hai loại CTĐ như vậy. Các khái niệm thích hợp để CTĐ như QT xử lý truyền thông tuần tự, lời gọi thủ tục từ xa và cuộc hẹn sẽ được mô tả.

CTĐ không đồng bộ (dị bộ)

Tuy trong CTĐ không chia xẻ biến nhưng các kênh truyền thông vẫn được chia xẻ. Vì vậy hoạt động nhận kết khối từ kênh truyền thông là tương đương với việc thực hiện toán tử P ở semaphore và gửi không kết khối là tương đương với toán tử V. CTĐ dị bộ đơn giản là việc mở rộng khái niệm semaphore cho hệ thống phân tán. Ở đây giả thiết rằng ***kênh có bộ đệm không hạn chế***. Việc đồng bộ CTĐ dị bộ cũng cần cấu trúc như semaphore, khi các kênh truyền thông có thể được chỉ ra trong một ngôn ngữ và được hỗ trợ bằng HĐH. Hình 3.18 minh chứng cách dùng CTĐ dị bộ cho loại trừ ràng buộc. Phục vụ kênh đại diện cho HĐH hỗ trợ cho các kênh logic. Nó tạo một kênh logic cho mục đích đồng bộ và khởi tạo kênh này để chứa TĐ. Nội dung TĐ ở trong kênh là không quan trọng đối với việc loại trừ ràng buộc. Các ví dụ tốt cho đồng bộ CTĐ là sử dụng ống dẫn (pipe) và (socket). Phục vụ kênh cũng tổ chức việc xếp hàng đợi các TĐ và xử lý khối trên kênh.

```
Process Pi                Channel phôc vô          Process Pj
Begin                     Begin                    Begin
    Receive(channel);         Create channel;          Receive(channel);
    Critical section;         Send(channel);           Critical section;
    Send(channel);            Manage channel;          Send(channel);
End;                      End;                     End;
```

Hình 3.18. Loại trừ ràng buộc dùng trong CTĐ dị bộ.

Chuyển thông điệp đồng bộ

CTĐ đồng bộ thừa nhận gửi kết khối và nhận kết khối. Điều này cần thiết khi ***không có bộ đệm các TĐ trên kênh truyền thông***. *Gửi* phải đợi cho tới khi có một QT nhận

tương ứng cùng diễn ra và *nhận* cũng phải đợi một QT gửi tương ứng. Có nghĩa bất cứ một QT nào đến trước phải đợi QT khác và việc đợi này là đối xứng. Cơ cấu này cho phép hai QT sau khi đối sánh cặp gửi và nhận thì kết nối và trao đổi dữ liệu tại một điểm đồng bộ và sau khi hoàn thành lại tiếp tục thực hiện hoạt động riêng của mình. Điểm đồng bộ như vậy được gọi là *cuộc hẹn giữa gửi và nhận*. Cuộc hẹn là một khái niệm có ích trong hệ thống máy tính cũng như trong đời sống hàng ngày. Ví dụ, bên ngoài sân vận động trước giờ bắt đầu của trận bóng, dễ dàng bắt gặp những người hoặc là cầm vé để bán hoặc là giơ những ngón tay để số hiệu vé họ cần mua. Cuộc hẹn giữa người bán và người mua ở đây là *dị bộ và đối xứng*.

Giải pháp loại trừ ràng buộc sử dụng CTĐ đồng bộ được mô tả ở hình 3.19. QT phục vụ semaphore bắt đầu bằng cuộc hẹn với hoặc Pi hay Pj; sau đó, cho phép QT được hẹn bước vào khoảng tới hạn, phục vụ ghi nhận id của QT xử lý và chờ đợi cuộc hẹn chỉ QT thực hiện hiệu quả của toán tử V. Điều đó hoàn thành một chu trình thực hiện loại trừ của khoảng tới hạn và phục vụ lặp lại việc chấp nhận một cuộc hẹn khác. Cũng như trên, nội dung TĐ là không quan trọng. Điều quan trọng hơn là cách gửi và nhận có thích hợp với nhau không ?

Cần một phương pháp để đối sánh tên gọi của hai cuộc hẹn bằng cách dùng các tên thủ tục. Dưới đây mô tả ba đồng bộ quan trọng theo tiếp cận CTĐ đồng bộ.

Process Pi	Semaphore phục vụ	Process Pj
Begin	loop	**begin**
Send(sem, msg);	Receive(pid, msg);	Send(sem, msg);
Critical section;	Send(pid, msg);	Critical section;
Receive(sem, msg);	**End**;	Receive(sem, msg);
End;		**end**

Hình 3.19. Loại trừ ràng buộc được dùng trong CTĐ đồng bộ

QT truyền thông tuần tự: Một tiếp cận Vào/Ra

QT truyền thông tuần tự (*CSP*) là đảm bảo ngôn ngữ đầu tiên định vị vấn đề đồng bộ trong hệ phân tán. Nó dùng các cuộc hẹn đầu vào/đầu ra để đạt được sự đồng bộ CTĐ. Đầu vào/đầu ra là một dạng của truyền thông TĐ. QT gửi P đưa một lệnh ra ***Q! exp*** đến QT nhận Q và QT Q cần có một lệnh vào tương ứng ***P? var***. Cuộc hẹn lệnh vào/lệnh ra được nối với nhau thông qua tên gọi của QT cần kết nối (P chỉ ra Q và Q chỉ ra P). Điều này tương đương với phép gán từ xa thực hiện việc gán giá trị ***exp*** từ một QT cho biến ***var*** của một QT khác. Việc trao đổi TĐ giữa các QT vào / ra là đồng bộ nên đạt được sự đồng bộ giữa hai QT.

Sử dụng trực tiếp tên QT khi truyền thông trong CSP có một vài hạn chế. Điều này có thể minh hoạ bằng thi hành bài toán đọc/ghi. Trước hết là tính không hợp lý khi yêu cầu QT đọc phải gửi TĐ cho QT ghi hoặc ngược lại bởi vì chúng được thực hiện một cách độc lập và không biết sự tồn tại của các QT khác. Thứ hai, chúng truyền thông trực tiếp với nhau là không cần thiết. Vì vậy chúng ta cần QT phục vụ nhận những yêu cầu từ QT đọc/QT ghi và qui định những luật cho Đọc đồng thời / Ghi độc quyền. Trong CSP đọc và ghi chỉ có một kiểu lệnh truyền thông với phục vụ S (ví dụ : ***S! req*** trong đó req là request message). Phục vụ sẽ thực hiện trao đổi đồng bộ bằng cách R? msg hoặc W? msg với msg là nội dung của TĐ nhận được còn R và W tương ứng là QT đọc và QT ghi, nhằm hỗ trợ việc không phải qui định cuộc hẹn giữa QT đọc và QT ghi. CSP đưa ra lệnh *alternative*, mỗi lênh này bao gồm một số lệnh *guarded*. Khi đưa

vào một lệnh *alternative* thì tất cả điều kiện của các lệnh *guarded* đều phải kiểm tra. Chỉ có một lệnh có điều kiện đúng được lựa chọn để thực hiện. Việc chọn các lệnh *guarded* ở trong lệnh *alternative* không được xác định trước. Tính không xác định của chương trình tuần tự là một mở rộng mơ ước đối với lập trình trong hệ phân tán. Một câu lệnh vào của CSP có điều kiên đúng khi câu lệnh ra tương ứng của nó được dùng.

Đối với vấn đề đọc/ghi, phục vụ cần một lệnh lựa chọn cho phép gặp QT đọc hoặc QT ghi. Vấn đề đầu tiên là phục vụ phải biết tên của tất cả các QT đọc và ghi tại thời điểm sinh mã để cung cấp cho QT điều khiển. Thứ hai, để sử dụng tên QT cho việc giao tiếp thì chỉ có một QT được yêu cầu phục vụ. Thao tác đọc và ghi thực sự không thể được thực hiên bằng QT người dùng mà cần được chuyển dưới dang mã của phục vụ. Chuyển tài nguyên đối tượng vào tài nguyên phục vụ là điều tốt vì nó cung cấp độ trong suốt cao hơn. Việc đọc đồng thời có thể được thực hiện bằng cách sử dụng câu lệnh song song CSP để tạo nên luồng QT đồng thời. Tuy nhiên việc đồng bộ giữa các quá QT vẫn cần được hoàn thiện trong phục vụ đa luồng. Điều này có nghĩa là chúng ta đưa trách nhiệm đồng bộ các luồng QT trong phục vụ. Những câu lệnh vào và ra đồng bộ chỉ đáp ứng cho mục đích truyền thông hỏi và đáp. Bỏ qua giải pháp bổ sung đặc tính đồng bộ cho luồng thì không có cách nào khác cho vấn đề đọc/ghi trong CSP. Khó khăn này có thể giảm bớt nếu mở rộng khái niệm vào/ra đồng bộ tới các thủ tục. Điều này dẫn đến sự phát triển của ***lời gọi thủ tục từ xa*** và những cuộc hẹn Ada.

Lời gọi thủ tục từ xa

Khái niệm vào/ra đồng bộ trong CSP cung cấp việc kết khối và trao đổi dữ liệu giữa các QT gửi và nhận. Lời gọi thủ tục có những đặc trưng tương tự mà theo đó thủ tục gọi được kết khối cho đến khi thủ tục được gọi hoàn thiện và dữ liệu được chuyển giao giữa thủ tục gọi và thủ tục được gọi. Truyền thông từ xa có thể dùng khuôn dạng lời gọi thủ tục giao tiếp và được thực hiện thực sự bởi CTĐ. Trong xử lý truyền thông, dùng lời gọi thủ tục thay cho tên QT đem lại hiệu quả hơn. Và kiểu lời gọi từ xa cũng có đặc tính trong suốt khi CTĐ được che dấu đối với người dùng.

Lời gọi thủ tục từ xa là truyền thông Client/Server. Tài nguyên của phục vụ được đại diện bằng tập hợp các thủ tục và chúng được dùng khi các khách từ xa dùng các thủ tục giao tiếp. Phục vụ cũng hoạt động giống như monitor. Hai phương pháp RPC và cuộc hẹn đều CTĐ đồng bộ tuy nhiên sự khác nhau giữa chúng là sự phân biệt phục vụ và các khách của nó. Các phục vụ trong RPC là bị động, chúng đưa ra các thủ tục của chúng và chờ đợi các khách dùng các thủ tục đó. Còn phục vụ trong cuộc hẹn thì ngược lại, chúng cố gắng tạo ra cuộc hẹn cho các yêu cầu của khách. Chính vì thế chúng ta có thể coi RPC là giao tiếp không đối xứng, còn cuộc hẹn là giao tiếp đối xứng. Và RPC được dùng như giao thức truyền thông còn cuộc hẹn dùng cho đồng bộ và đó là cơ sở cho việc thực hiện cuộc hẹn từ xa trong các hệ thống phân tán.

Cuộc hẹn Ada: Một tiếp cận lời gọi thủ tục từ xa và đối xứng

Dù không phải được thiết kế cho các cuộc hẹn từ xa trong môi trường mạng phân tán nhưng những cuộc hẹn Ada vẫn là ví dụ tốt cho việc dùng khái niệm lời gọi thủ tục và cung cấp cấu trúc ngôn ngữ không xác định cho QT đồng thời. Các thủ tục trong RPC là tĩnh và cần được kích hoạt bằng lời gọi. Để dùng được các thủ tục trong cuộc hẹn, chúng bắt buộc có được tính sẵn sàng động để luôn sẵn sàng đáp ứng các lời gọi. Ngôn ngữ Ada đưa ra những câu lệnh chấp nhận cho mục đích này. Một câu lệnh chấp nhận có một bộ phận vào được cho dưới dạng dãy câu lệnh xác định thủ tục. Bộ phận vào chứa tên điểm vào và các tham biến hình thức. Việc yêu cầu như cuộc hẹn với câu lệnh chấp nhận tương ứng với cùng tên điểm vào thủ tục. Việc kết khối là đối xứng theo cách QT đầu tiên xuất hiện (lời gọi khác) chờ bộ đếm của nó xuất hiện. Điểm hẹn này

cung cấp điểm đồng bộ giữa hai QT. Thực hiện câu lệnh chấp nhận bao gồm việc trao đổi tham số và phân tử của nó. Mã lệnh không loại trừ thực sự được ghi ngoài câu lệnh chấp nhận và có thể thực hiện đồng thời.

Nếu các QT có cấu trúc khách và phục vụ thì các câu lệnh chấp nhận thường xuất hiện trong QT phục vụ. Các khách yêu cầu phục vụ bằng cách gọi những câu lệnh chấp nhận của phục vụ một cách trực tiếp. Do phục vụ cần hẹn với nhiều khách một cách không định trước, thì câu lệnh lựa chọn (giống như lệnh *alternative* trong CSP) được bổ sung vào ngôn ngữ Ada. Câu lệnh *select alternative* có dạng của lệnh *guarded* theo nghĩa cho biết điều kiện đối với lệnh chấp nhận. Khi một lệnh chọn được đưa vào, thì tât cả các điều kiên đảm bảo được kiểm tra và đánh dấu nếu như điều kiện đó đúng. Một trong những câu lênh đã đánh dấu mà chưa giải quyết sẽ được chọn thực hiện theo cách không định trước. Hình 3.18 minh chứng cách dùng câu lệnh chấp nhận và câu lệnh chọn cho vấn đề ưu tiên QT đọc yếu. Nhiệm vụ của rw (phục vụ) sẽ hẹn với QT đọc và QT ghi một cách không định trước. Các QT đọc/ghi tương tác với phục vụ qua lời gọi bài toán thích hợp với điểm vào giống như lời gọi thủ tục thông thường.

```
task rw is
        entry       strartread;
        entry endread;
        entry startwrite;
        entry endwrite;
end;

task body rw is
        rc: integer := 0;
  busy: boolean := false;
begin
loop
        select
                when busy = false →
                accept startread do rc := rc + 1 end;
        or
                →
                accept endread do rc := rc - 1 end;
        or
                when rc = 0 and busy = false →
                accept startwrite do busy := true end;
        or
                →
            accept endwrite do busy := false end;
end loop
end;
```

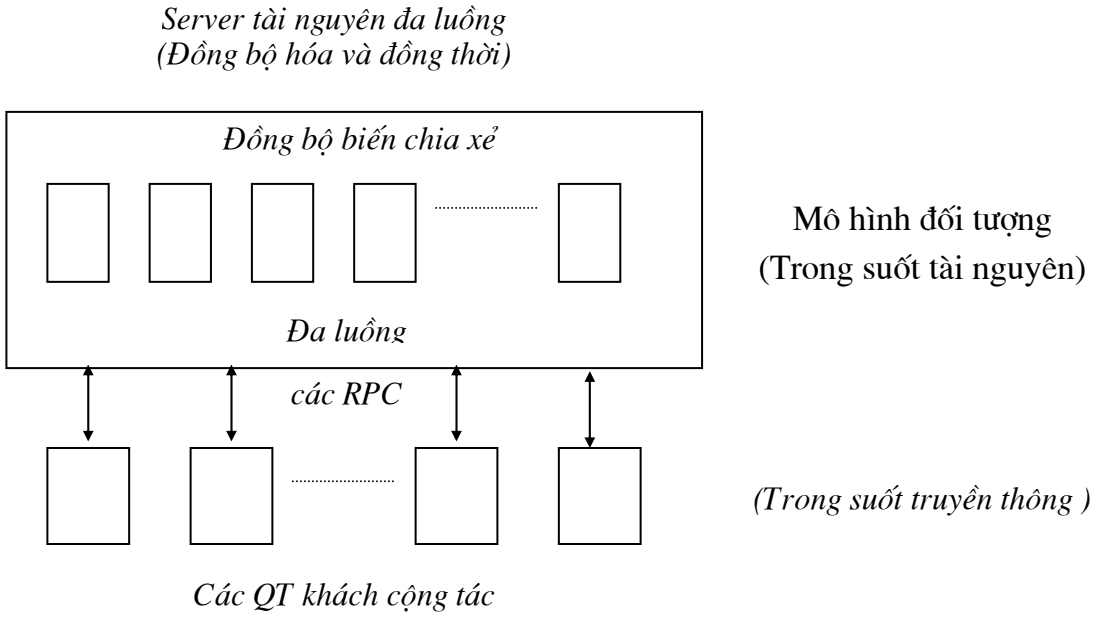
Hình 3.20. Giải pháp cuộc hẹn Ada cho vấn đề ưu tiên QT đọc yếu

Kết hợp câu lệnh chấp nhận và câu lệnh chọn cung cấp việc loại trừ ràng buộc và đồng bộ cho đọc/ghi. Đọc/Ghi thực sự cũng có thể được nhúng trong phục vụ đồng bộ. Đọc đồng thời vẫn có thể dùng câu lệnh ***accept-startread*** bằng việc ***fork*** QT hoặc luồng khác. Trong hệ phân tán, tên điểm vào có thể được truyền đi và cuộc hẹn của thủ tục được thực hiện bằng lời gọi thủ tục từ xa.

3.6 Mô hình đối tượng các phục vụ tài nguyên

Nhu cầu đồng bộ xuất hiện từ việc chia xẻ tài nguyên. Để làm tăng hiệu quả việc sử dụng tài nguyên cần giao trách nhiệm cho phục vụ quản lý tài nguyên. Sử dụng khái niệm hướng đối tượng đối với tài nguyên sẽ cung cấp sự trong suốt tài nguyên tới các khách. Một tài nguyên được coi là một đối tượng ảo và được trình bày dưới dạng một tập các thao tác chính xác được khách gọi. Đồng bộ và điều khiển đồng thời giữa các đối tượng phục vụ là hoàn toàn trong suốt với các khách. Hơn nữa truyền thông giữa các khách và phục vụ đối tượng có thể trong suốt bằng cách dùng các lời gọi thủ tục từ xa. Sự đồng thời trong phục vụ có thể được thực hiện tốt bằng cách dùng đa luồng cho phép đáp ứng đồng thời các yêu cầu của nhiều khách. Do các luồng chia xẻ vùng địa chỉ chung, phương pháp đồng bộ biến chia xẻ được sử dụng để phối hợp các luồng.

Trong RPC và cuộc hẹn đã thảo luận nhiều về việc đồng bộ CTĐ. Trong một vài ứng dụng, các khách có thể ưa thích gửi các yêu cầu dị bộ cho phục vụ. Một yêu cầu không cần sự đáp lại nên là dị bộ. Điều này cũng có thể xảy ra khi cần lời dáp cho một câu hỏi song để hiệu quả hơn, QT khách thực hiện một công việc khác thay vì bị kết khối. Có hai phương pháp thi hành việc CTĐ không đồng bộ với phục vụ: một là định nghĩa RPC dị bộ mới, hai là ứng với mỗi RPC tạo một luồng của QT khách. Cả hai phương pháp đều cần tới cơ chế ngôn ngữ, với cơ chế đó QT có thể biết được việc hoàn thành và lấy được kết quả của các RPC.

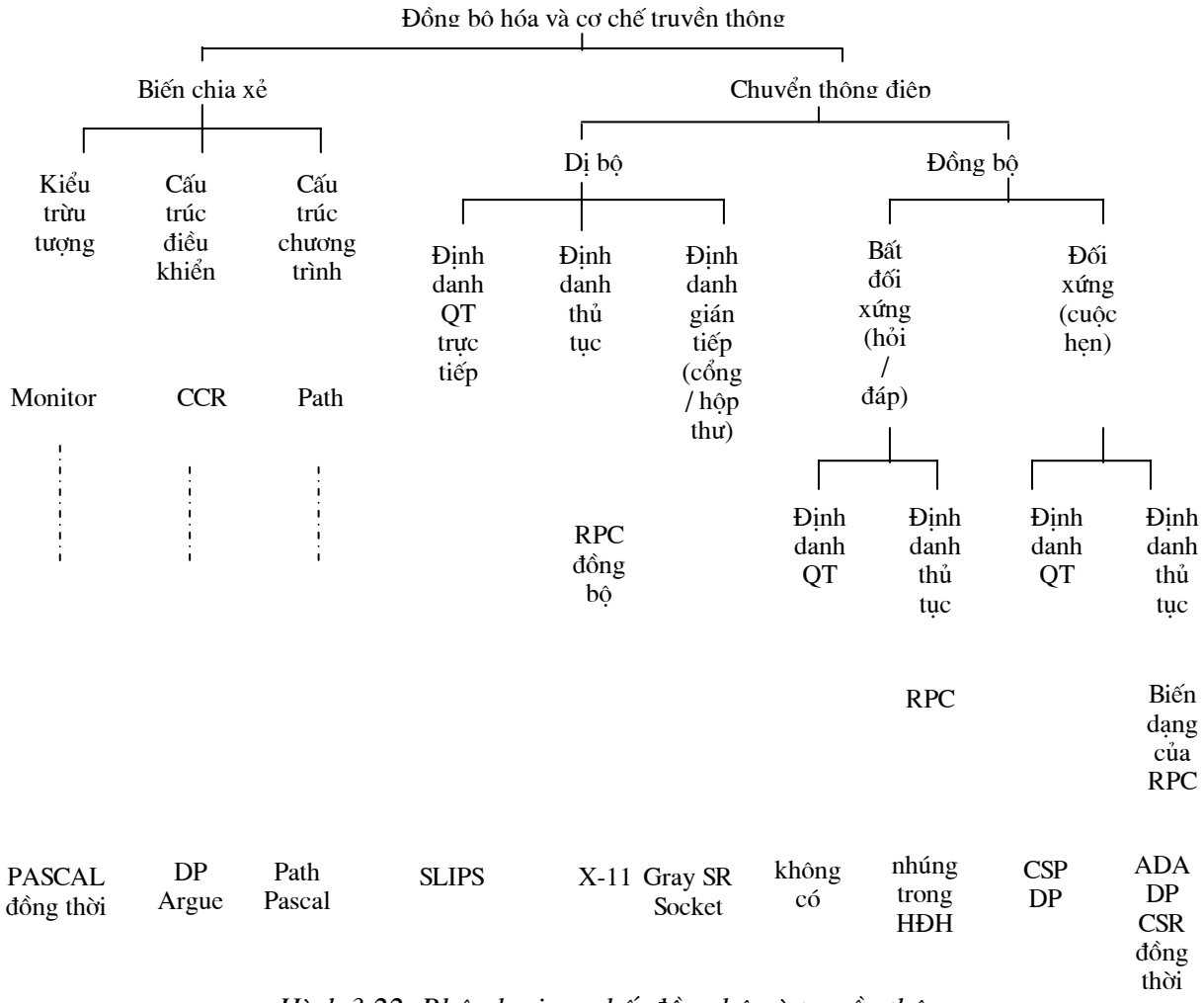


Hình 3.21. Cấu trúc của phục vụ tài nguyên phục vụ đối tượng

3.7. Ngôn ngữ lập trình đồng thời

Ngôn ngữ lập trình đồng thời là một hệ chương trình hỗ trợ việc đặc tả tính đồng thời, sự đồng bộ, việc truyền thông trong tương tác giữa các QT đồng thời. Sự thi hành của ngôn ngữ lập trình đồng thời được dành cho HĐH phân tán hơn là chương trình dịch như kiểu các ngôn ngữ tuần tự. Điều đó do tính đồng thời, đồng bộ và truyền thông QT là một vấn đề thời gian chạy. Ngôn ngữ lập trình đồng thời có thể được xem như là sự mở rộng từ ngôn ngữ tuần tự đã tồn tại. Chúng ta xem xét cơ chế đồng bộ theo minh họa trong hình 3.22.

Cơ chế đồng bộ được phân thành hai loại: Biến chia xẻ hoặc CTĐ. *Monitor* là một ví dụ thông dụng nhất cho khái niệm đồng bộ biến chia xẻ cho những ngôn ngữ đồng thời. Sự thể hiện trừu tượng nhờ khái niệm monitor làm cho nó trở thành mô hình thích hợp đối với đối tượng dữ liệu và phục vụ tài nguyên. Những ngôn ngữ cổ điển sử dụng monitor là Concurent Pascal, Modula và Turing Plus. Concurent Pascal là ngôn ngữ đầu tiên hỗ trợ monitor. Modula là một ngôn ngữ được phát triển với mục đích nhấn mạnh tầm quan trọng của modun chương trình. Nhờ bổ sung môdun giao tiếp hỗ trợ monitor, Modula sử dụng môđun thiết bị nhằm thi hành các I/O trừu tượng cho phép mềm dẻo hơn khi giao tiếp với nhân. Turing là một ngôn ngữ tuần tự được thiết kế phục vụ chỉ dẫn tới các công thức toán học mạnh. Turing Plus là một mở rộng của Turing nhằm hỗ trợ monitor cho lập trình đồng thời. Tồn tại nhiều ngôn ngữ đồng thời dựa trên monitor và việc định nghĩa monitor cũng rất đa dạng. Điều khác nhau chính của chúng là ngữ nghĩa là toán tử ***signal*** cùng lời gọi monitor lồng nhau và quy định phạm vi của các biến.



Hình 3.22. Phân loại cơ chế đồng bộ và truyền thông

Một ngôn ngữ đồng thời danh tiếng khác sử dụng biến chung đồng bộ đồng thời là Path Pascal, một biến thể của khoảng tới hạn điều kiện - CCR trong QT phân tán - DP (*Distributed Processes*) và Argus. Biểu thức Path là cách đặc tả ngữ nghĩa cho một tính toán đồng thời. Tuy nhiên, biểu thức Path đơn giản không hiệu quả trong các mô hình cộng tác QT trong đó đòi hỏi thông tin trạng thái về tài nguyên chia xẻ. Biểu thức Path mở rộng được đề xuất để thể hiện việc mở rộng mô hình kiểu đó nhưng khi mở rộng như vậy thì sự đẹp đẽ ban đầu lại bị mất đi. Khoảng tới hạn điều kiện CCR dễ dàng được thi hành. Đánh giá các điều kiện nhằm tách khối các QT là rất đáng giá. Điều này có thể được giải quyết bằng ***wait*** và ***signal*** hiển theo điều kiện tương tự như ***biến điều kiện*** trong tiếp cận monitor.

Hai phương pháp đồng bộ CTĐ đồng bộ và dị bộ được sử dụng trong nhiều ngôn ngữ đồng thời. Việc nhận hoặc gửi TĐ có thể được xác định bằng cách chỉ trực tiếp tên nguồn và tên đích của TĐ. Tuy nhiên, như đã được diễn giải ở trên, cách chỉ tên như vậy là không thực tế. Mềm dẻo hơn, QT TT là không trực tiếp mà được thực hiện bằng cách qua kênh định danh. Cổng (*Port*) và hộp thư (*Mailbox*) là hai kênh chuyển định danh. Cổng cho phép truyền thông nhiều-một. Hộp thư cho phép truyền thông nhiều-nhiều. Sử dụng cổng hoặc hộp thư không giới hạn việc cung cấp gửi TĐ dị bộ. GYPSY là ngôn ngữ bậc cao đầu tiên sử dụng hộp thư để gửi các TĐ dị bộ. PLITS là ngôn ngữ sử dụng việc CTĐ dị bộ nhưng định danh trực tiếp các môđun QT. Vấn đề định tên trực tiếp trong PLITS được làm nhẹ bớt bằng việc bổ sung khóa TĐ và những phép toán cho phép loại bỏ những thông tin được bổ sung từ bên gửi. Một QT nhận có thể

nhận mọi TĐ từ bất cứ nơi gửi nào hay bất cứ TĐ nào với cùng một khoá thẻ (tagged key) thực sự. Hợp lý hơn nếu QT nhận trở thành một phục vụ dành cho khách phức mà không cần biết tên của các khách. Nói thêm là kênh TT phức giữa khách và phục vụ được thiết lập khi dùng các khóa khác nhau đối với TĐ.

Chính vì vậy, bên nhận có thể được coi như là một phục vụ dành cho nhiều khách mà không cần biết tên của các khách. Sử dụng khoá thẻ khác nhau có thể tạo nên những kênh truyền khác nhau giữa phục vụ và khách.

Thiên hướng trên được sử dụng trong ngôn ngữ C đồng thời để chuyển đồng bộ những TĐ dị bộ. Có thể đặt tên cho kênh bằng cách sử dụng những cấu trúc ngôn ngữ bậc cao qua các lời gọi thủ tục và được gọi là RPC dị bộ.

CTĐ đồng bộ có thể đối xứng hoặc không đối xứng, phụ thuộc vào kịch bản truyền thông. CTĐ đồng bộ không đối xứng giả thiết hai chủ thể truyền thông có quan hệ chủ/tớ - *master*/*slave* (chính-phụ - *primary*/*secondary*). Đối tượng chính phát yêu cầu và đối tượng phụ đáp ứng câu trả lời cho yêu cầu đó. Khi thi hành trong mô hình phục vụ/client thì phục vụ đóng vai trò như là đối tượng nhận yêu cầu phục vụ. CTĐ đồng bộ đối xứng là khái niệm cuộc hẹn khi các QT truyền thông đang cố gắng tích cực đồng bộ với một QT khác. Tương tự, cả đồng bộ hỏi/đáp và cuộc hẹn đều yêu cầu thiết lập kênh truyền thông bằng tên QT hoặc lời gọi thủ tục. Cổng và hộp thư không thể áp dụng ở đây vì chúng được dùng cho gửi dị bộ. RPC quy ước dùng CTĐ đồng bộ không đối xứng và các lời gọi thủ tục được hợp thể hoá trong một số ngôn ngữ đồng thời như DP, Argus và Mesa. Các ngôn ngữ này cũng bổ sung đồng bộ CCR hoặc monitor biến chia xẻ. Do RPC là phương pháp truyền thông quan trọng trong hệ phân tán nên nó được tích hợp như một gói phần mềm trong hầu hết HĐH hiện nay.

CTĐ đồng bộ đối xứng dùng cuộc hẹn. CSP dùng cách đánh tên QT cho cuộc hẹn. DP mở rộng thêm tên thủ tục trong QT, nơi cuộc hẹn có thể xẩy ra tại các điểm khác nhau. Hiệu quả sử dụng cuộc hẹn theo kiểu RPC tìm thấy trong Ada. Thi hành Ada cho cuộc hẹn RPC là đúng hoàn toàn. Lệnh ***seclect*** hỗ trợ lời gọi cuộc hẹn không định trước (truyền thông chọn lọc). Bổ sung thêm những chức năng về quá hạn (time-out) và điểm vào thủ tục cho vector ngắt trong những lệnh ***accept***. Ada trở thành thích hợp cho lập trình hệ thống và lập trình thời gian thực. Hình 3.20 trình bày sự phân loại các mô tả trên đây về đồng bộ và truyền thông ngôn ngữ đồng thời. Nhiều ngôn ngữ đồng thời dùng cách tổ hợp các cơ chế. ví dụ, ngôn ngữ SR (Synchronizing Resources) có hâu hết các cấu trúc cần thiết cho lập trình đồng thời. SR dùng sự trừu tượng tài nguyên, PRC dị bộ và RPC cuộc hẹn, truyền thông chọn lọc, mô đun hóa phần cứng và mô đun hóa ngắt.

Đi tới các tiệm cận ngôn ngữ đồng thời khác ngoài hướng mở rộng đơn giản các ngôn ngữ tuần tự cũng rất có giá trị. Bàn luận trên đây chỉ ra rằng quản lý QT đồng thời, đồng bộ cũng như truyền thông thực ra là những vấn đề trực giao với khía cạnh tính toán của ngôn ngữ tuần tự. Thực ra cần tìm một ngôn ngữ phối hợp theo phương thức đơn giản và hiệu quả để xây dựng chương trình đồng thời bằng cách kết nối một vài thực thể truyền thông cơ sở lại và cung cấp ý nghĩa truyền thông và đồng bộ nhau cho chúng. Các thực thể truyền thông là QT, tài nguyên, hoặc cả hai hoặc là đối tượng trừu tượng. Với giả thiết CTĐ cho truyền thông và đồng bộ QT, thì các vấn đề mấu chốt chỉ là thực thể được mô hình hóa như thế nào, các chương trình đồng thời được giải quyết ra sao và các kênh truyền thông giữa các thực thể sẽ được đặt tên lôgic và quản lý như thế nào ? Một ngôn ngữ đồng thời mới (chi tiết hơn, hệ lập trình đồng thời) có thể được xây dựng dựa trên những mô hình tính toán và truyền thông. Đoạn dưới đây giới thiệu và so sánh ba hệ thống, được thiết kế theo dòng tiếp cận này: Occam, SR và Linda.

Occam tiến hóa từ nhiều ý tưởng trong CSP. Nó được dùng rộng rãi để lập trình đồng thời trong các hệ thống Transputer và xử lý tín hiệu số DSP (*Digital Signal Processing*). Transputer và DSP là các máy tính đơn chíp với bộ nhớ trong cục bộ và tuyến truyền thông nhanh. Nó có thể cấu hình để ánh xạ các QT đồng thời tới xử lý song song hiệu quả với việc truyền thông dữ liệu thường xuyên giữa các bộ xử lý nhờ tuyến tốc độ cao. Do các bộ xử lý đơn chíp này có bộ nhớ cục bộ hạn chế nên OCCAM giả thiết hạt QT nhỏ. Mỗi lệnh sai khiến (gán, vào và ra) được coi như những QT nguyên thủy. Lệnh được nhóm với nhau nhờ lệnh ***contructors*** thành lệnh hợp thành theo một trong ba kiểu thực hiện: thực hiện tuần tự các lệnh (cấu trúc SEQ), thực hiện song song các lệnh (cấu trúc PAR), thực hiện dị bộ và không định trước cấu trúc ALT tương tự như lệnh ***alternative*** trong CSP. Hai cấu trúc bổ sung, IF và WHILE cần đến để điều chỉnh dòng tuần tự thực hiện cấu trúc. Cấu trúc có thể cộng tác với các biến cục bộ và có thể gộp trong cấu trúc khác. Mặc dù lệnh là QT nguyên thủy thực hiện được, cấu trúc là đơn vị QT lập lịch chuẩn. Không có sự chia sẽ biến toàn cục giữa các cấu trúc khác nhau. Đồng bộ được thực hiện bằng cách sử dụng khái niệm cuộc hẹn của lệnh input/output trong CSP, ngoại trừ tên các kênh truyền thông dùng các tên kênh toàn cục khai báo tường minh. Sử dụng cấu trúc để dàn xếp các chương trình đồng thời và kênh toàn cục để truyền thông giữa các cấu trúc làm cho trình biên dịch dễ tạo ra mã xếp lịch các QT tới bộ xử lý.

Trong **SR**, chương trình đồng thời là một tập các tài nguyên, khác với cách nhìn QT của Occam. Tài nguyên được trừu tượng như một môđun gồm khai báo và thân. Khai báo đặc tả những thực thể nhập đối với các tài nguyên khác và thực thể xuất đối với các phép toán trên chính tài nguyên này. Thân chứa một phần khởi tạo, một hoặc nhiều QT và một mã kết thúc. Các QT trong một tài nguyên có thể đồng thời hoặc tương tác nhau nhờ biến chia sẻ trong nguốn đó. Các tài nguyên khác nhau tương tác nhờ các ***operations*** tương tự như thực thể thủ tục trong Ada. Thực thể ***operations*** hoặc là QT hoặc là thủ tục.

Minh họa cho một tài nguyên được dẫn ra qua việc thực hiện lệnh CREATE tên_tài_nguyên trả lại năng lực để truyền thông sau này với các phép toán tài nguyên. Phép toán tài nguyên được dẫn ra lênh CALL đồng bộ hoặc lệnh SEND không đồng bộ tới tài nguyên khi dùng năng lực tài nguyên. Trong Occam, năng lực là mềm dẻo hơn so với kênh toàn cục. Chúng được tạo ra một cách động và có thể được truyền như một biến. Thêm vào nữa, năng lực có thể trình bày đa thể hiện của một tài nguyên và đạt được cả khái niệm đặt tên kênh và điều khiển truy nhập. Cuộc hẹn được đặc tả bằng lệnh nhập (***in***) với điểm vào phép toán có lựa chọn (***ana***) để hỗ trợ truyền thông chọn lọc và lựa chọn (***by***) để lập lịch các yêu cầu sắp giải quyết. SR dùng trừu tượng dữ liệu và hỗ trợ hầu hết mọi kiểu đồng bộ CTĐ và chia sẻ biến.

LINDA khác biệt với Occam hoặc SR. Nó không là một ngôn ngữ lập trình nhưng một mô hình dữ liệu chia xẻ duy nhất có thể được tích hợp với bất kỳ ngôn ngữ lập trình nào để hỗ trợ cộng tác QT trong lập trình song song. QT và dữ liệu chia xẻ trong mô hình Linda được trình bày đồng nhất như một tập không sắp xếp của các bộ (tuple) mà mỗi bộ có dạng ***t*** = ("tag", value), trong đó ***tag*** là tên kí hiệu của bộ dữ liệu còn ***value*** là danh sách giá trị có kiểu phù hợp với bộ đó. QT là bộ hoạt động (*active tuple*) còn bộ dữ liệu là bộ thụ động (*pass tuple*). Các bộ thuộc lớp không gian bộ (TS), một vùng bộ nhớ lôgic địa chỉ hóa được nội dung chia xẻ, có thể phân tán một cách vật lý để trình bày cấu trúc dữ liệu phân tán trong không gian bài toán. Linda cung cấp ba nguyên thủy cơ sở để truy nhập bộ. Hai nguyên thủy ***in***(***s***) và ***out***(***s***) là nhận kết khối từ TS và gửi không kết khối tới TS, ở đây, ***s*** là một mẫu hoặc một anti-tuple có dạng ***s*** =

("tag", Actual, formats). ***in***(***s***) làm phù hợp với mẫu ***s*** dựa vào bộ ***t*** với thẻ tag và thực tế Actual. Nếu phù hợp, nghi thức trong ***s*** được gán bằng các giá trị tương ứng trong ***t*** và QT gọi được tiếp tục. Sau đó, bộ này bị xoá khỏi TS. Trong trường hợp phù hợp bội (có nhiều hơn một bộ phù hợp) thì chỉ có duy nhất một bộ được chọn một cách tuỳ biến. Phép toán ***in*** bị kết khối theo nghĩa QT gọi bị ngừng lại cho đến khi có được sự phù hợp. Phép toán ***out*** đơn giản tính giá trị của biểu thức trong ***s*** và đưa vào trong TS. Cập nhật bộ được thực hiện này sau khi có sự phù hợp ***in*** ngay sau một nguyên thủy ***out***. Nguyên thủy thứ ba là ***rd*** được dùng để đánh giá bộ. Nguyên thủy này tương tự như ***in***, ngoại trừ còn phải làm phù hợp với các bộ còn lại trong TS. Các phiên bản ***in*** và ***rd*** không kết khối cũng được Linda hỗ trợ.

Linda có một nguyên thủy bổ sung *eval(s)* tạo ra QT. Phép toán *eval* khởi tạo (*fork*) một QT mới nhằm đánh giá tất cả các biểu thức trong *s*. Các biểu thức này thường chứa các thủ tục. *eval* tương tự như *out*, ngoại trừ việc *s* được đưa vào TS trước khi đánh giá nó. Khi QT được hoàn thành (tức mọi biểu thức đã được tính), *s* cùng các giá trị kết quả thành một bộ dữ liệu thụ động. Linda là cách trừu tượng nhất để mô hình hóa cộng tác quá trình. Do chỉ có một số rất ít các phép toán nguyên thủy, Linda có thể được gắn vào ngôn ngữ cơ sở, chẳng hạn C. C-Lind là thi hành của mô hình Linda trong C.

Hình 3.23 mô tả những nét khác nhau cơ sở giữa Occam, SR và Linda theo các phương diện là khái niệm hệ thống cơ sở, mô hình dữ liệu căn bản, cách thức đặt tên cho kênh truyền thông.

	Hệ thống	*Mô hình đối tượng*	*Đặt tên kênh*
Occam	Ngôn ngữ lập trình đồng thời	Quá trình	Kênh toàn cục tĩnh
SR	Ngôn ngữ lập trình đồng thời	Tài nguyên	Năng lực động
Linda	Ngôn ngữ lập trình đồng thời	Cấu trúc dữ liệu phân tán	Thẻ kết hợp

Hình 3.23 So sánh Occam, SR và Linda

3.8 Lập trình phân tán và lập trình trên mạng

Occam, RS và Linda được mô tả như những ngôn ngữ đồng thời hoặc những mô hình cho hệ thống chặt chẽ. Chúng không đáp ứng được cho hệ thống không chặt chẽ hoặc là mạng máy tính, nơi những vấn đề về khả năng trong suốt và khả năng tương tác các toán tử là cần được suy nghĩ. Trong hệ thống lỏng, sự thực hiện các hệ chương trình đồng thời cần được cung cấp sự truyền thông tin cậy, đối tượng riêng biệt, nhân bản dữ liệu và một hệ thống hỗn tạp. Hai ngôn ngữ lập trình ORCA và JAVA là ví dụ liên quan đến lập trình hệ phân tán và mạng.

3.8.1 ORCA

Orca là ngôn ngữ lập trình động thời dựa trên mô hình đối tượng chia sẻ logic cho hệ thống phân tán lỏng. Nó hỗ trợ việc thực hiện song song của QT trên các máy khác nhau nhờ truy nhập đồng thời đối tượng dữ liệu chia xẻ. Đối tượng dữ liệu chia xẻ là các thể hiện của kiểu dữ liệu trừu tượng được người dùng định nghĩa. Chúng có thể được phân tán vật lý hoặc được nhân bản để truy nhập cục bộ hiệu quả và thuận tiện nhưng chúng hoàn toàn "trong suốt" đối với chương trình ứng dụng nhờ thi hành của Orca.

QT Orca mỗi khi được khởi tạo, truyền thông xuyên qua các đối tượng chia xẻ khi dùng các phép toán người dùng định nghĩa trên các đối tượng đó. Ưu điểm của việc sử dụng biến chia xẻ cho TTLQT thì tương phản với CTĐ là hoàn toàn rành mạch. Truyền thông liên bộ xử lý là hoàn toàn "trong suốt" và các thông tin toàn cục có thể được chia xẻ trực tiếp. Viết một chương trình đồng thời là gần gũi với kiểu lập trình thông thường. Thêm nữa, nếu thể hiện cấu trúc dữ liệu phức tạp của đối tượng được phân tán vật lý, việc sử dụng CTĐ tường minh để truy cập vào từng phần của cấu trúc dữ liệu sẽ trở nên khó khăn nếu không quá dài dòng.

Khái niệm cốt lõi được dùng trong hệ thống Orca là đối tượng chia xẻ lôgic và kiểu dữ liệu trừu tượng. Khái niệm đối tượng chia xẻ lôgic (hoặc cấu trúc dữ liệu) đã từng biết trong Linda còn khái niệm kiểu dữ liệu trừu tượng có trong SR. Trong Linda đối tượng chai xẻ (bao gồm dữ liệu và QT) là các bộ trong không gian bộ. Phương thức truy nhập là phù hợp (địa chỉ hóa nội dung) nhưng ở mức thấp (nguyên thủy *in, out, rd*). Mặc dù về ngữ nghĩa thì khá đơn giản và đẹp đẽ, nhưng viết một chương trình với cấu trúc dữ liệu phức tạp trong Linda là không trực giác (nhúng trong những ngôn ngữ lập trình). Dùng kiểu dữ liệu trừu tượng trong Orca thì cho phép sự chặt chẽ kiểm tra kiểu và độ mềm dẻo của các phép toán trên đối tượng. SR là ngôn ngữ định kiểu mạnh và có tính trừu tượng kiểu dữ liệu giống như Orca. Tuy nhiên, do không có khái niệm đối tượng chia xẻ logic nên SR phải dựa vào một tập hợp lớn nguyên thủy đồng bộ và truyền thông (cuộc hẹn, CALL, SEND, năng lực và những đồng bộ biến chia xẻ khác). Mặt dù ngôn ngữ này khá dồi dào cho lập trình đồng thời, nhưng đồng bộ và truyền thông QT lại không trong suốt. Orca được xem như ngôn ngữ thoả hiệp tốt giữa Linda và RS.

QT đồng thời trong Orca được kích hoạt bằng cách tạo ra một QT mới trên một máy hoàn toàn xác định bởi lệnh fork:

fork Tên_QT (các tham số) { on [số Bộ XL]}

Tham số được truyền từ QT cha xuống QT con có thể là giá trị, đối tượng thường hoặc đối tượng chia xẻ. Giá trị và đối tượng thường là những dữ liệu cục bộ của QT. Đối tượng chia xẻ là toàn cục. Chúng được nhân bản tại mỗi bộ xử lý và cần một giao thức cập nhật nguyên tử để duy trì nhất quán của các bản sao của đối tượng chia xẻ.

Hai kiểu đồng bộ trên đối tượng được Orca cung cấp là loại trừ ràng buộc và cộng tác có điều kiện. Orca cho rằng tất cả các phép toán đều hoàn toàn cô lập trên đối tượng. Mỗi đối tượng được gắn một khóa để loại trừ ràng buộc. Khoá chỉ có hiệu lực tại mức đối tượng và được giới hạn với đối tượng đơn. Khoá chia xẻ được cung cấp để cho phép cùng một lúc có nhiều phép toán đọc đối tượng. Để cộng tác có điều kiện, Orca sử dụng lệnh an toàn (*Guarded)* tương tự như trong CSP:

Operation Op(parameters)	//***operation*** Tên_toán_tử(các_tham_số)
Guarded condition do statements;	//*Guarded* điều_kiện ***do*** các_câu_lệnh;
Guarded condition do statements;	// *Guarded* điều_kiện ***do*** các_câu_lệnh;

Lệnh trong toán tử chỉ được thực hiện khi điều_kiện trong *Guarded* là đúng. Nếu không thì QT đó bị chặn lại. Nếu có hơn một điều kiện đúng, thì chỉ một lệnh điều kiện được chọn động để cung cấp cách thức thực hiện không xác định. Một toán tử đối tượng trong một chương trình ứng dụng trên Orca được biên dịch thành nguyên thủy *invoke* (yêu cầu).

invoke (object, operation, parameters)

Các nguyên thủy *invoke* được bẫy tới hệ thống thời gian thực hiện Orca (Orca RTS: *Orca Run Time System*). RTS kiểm tra xem đối tượng có phải là ***chỉ đọc***. Nếu đúng, RTS đặt khoá chia xẻ cho bản sao địa phương, thực hiện QT đọc, sau đó lại mở khóa

đối tượng. Nếu không, RTS khởi tạo một TĐ quảng bá đến tất cả các vị trí của đối tượng chia xẻ, bao gồm chính nó, để cập nhật đối tượng và kết khối QT. Khi nhận TĐ quảng bá để ghi lên đối tượng, RTS đặt một khoá ghi lên đối tượng, thực hiện toán tử ghi và mở khoá đối tượng. Nếu thông điệp quảng bá hình thành cục bộ, nó sẽ kết khối QT.

RTS giả thiết truyền thông là tin cậy. Nó hỗ trợ tầng quảng bá tin cậy ngay dưới hệ thống thời gian chạy. Tầng quảng bá tin cậy đó có thể thi hành nhiều giao thức quảng bá theo những đòi hỏi ngữ nghĩa khác nhau. Giao thức quảng bá kỳ vọng đảm bảo rằng mọi phía đối tượng nhận được toàn bộ các TĐ quảng bá và tất cả các TĐ được phân phát theo đúng một thứ tự (tức là quảng bá được xem như là một nguyên tử). Thi hành giao thức quảng bá nguyên tử được trình bày ở chương sau, tuy vậy ở đây giới thiệu khái quát cách tiếp cận đơn giản trong Orca để thực hiện quảng bá nguyên tử. Khi quảng bá được RTS yêu cầu, nhân của RTS gửi một TĐ điểm-điểm đến lời gọi nhân đặc biệt *sequencer*. *Sequencer* gắn một số hiệu dãy tới yêu cầu và quảng bá TĐ bao gồm các số hiệu dãy tới mọi phía đối tượng nhân bản. Chi số dãy được mỗi nhân dùng để xác định thứ tự phân phát TĐ, kiểm tra TĐ bội, và yêu cầu *sequencer* chuyển lại một TĐ nếu như TĐ đó bị mất.

Đối tượng là đơn vị dữ liệu nền tảng trong Orca. Đối tượng được trình bày bằng một cấu trúc dữ liệu chẳng hạn như danh sách, cây hoặc đồ thị, thường được dùng bằng con trỏ trong ngôn ngữ lập trình quy ước. Con trỏ (pointer) là một địa chỉ máy. Truyền địa chỉ máy là kém ngữ nghĩa và có thể dẫn đến xung đột an ninh CTĐ ở HPT. Cơ chế cho phép thi hành và truyền cấu trúc dữ liệu phức tạp có sẵn để tạo ra mô hình đối tượng chia xẻ hữu dụng. Orca giải quyết vấn đề này bằng cách thay thế con trỏ bởi tên. Với mỗi đối tượng được RTS quản lý, mảng logic các dữ liệu có cấu trúc sẽ được duy trì cho đối tượng. Ví dụ, một cây nhị phân đơn giản *t* với 3 nút {A, B, C} và các liên kết trái và phải, được trình bày như sau:

t[1] = 6,A,8

t[6] = 0,B,0

t[8] = 0,C,0

Mỗi nút trong cấu trúc dữ liệu này được tạo động bởi nguyên thủy RTS *addnode*(*t*), trả lại tên *n* cho nút đó. Tên *n* được dùng như số hiệu của cấu trúc mảng để đặt tên cho nút cũng như để liên kết tới các nút khác. Tương tự cũng có một nguyên thuỷ xóa nút là *deletenote*(t,n). Chỉ dẫn tới nút đã bị xóa sẽ gặp lỗi thực hiện. Mảng đối tượng mang tính lôgic: định vị và giải phóng lưu giữ chúng được RTS quản lý động. Dùng nút đặt tên để thi hành cấu trúc dữ liệu đạt được hiệu lực tới con trỏ mà không cần địa chỉ máy. Truyền cấu trúc dữ liệu phức tạp trở thành chấp nhận được với việc trả thêm tổng phí trong hệ thống thời gian chạy.

3.8.2 Java

Mục tiêu nguyên thủy của Orca là hỗ trợ lập trình đồng thời trong hệ phân tán. Vấn đề thi hành chính yếu của nó là tính toán phân tán và tạo độ trong suốt truyền thông tới các QT cộng tác. Java được đưa ra theo một phối cảnh khác. Nó là ngôn ngữ lập trình và môi trường lập trình, nhằm đạt được khả năng cộng tác trong phát triển phần mềm mạng. Chúng ta có thể hình dung rằng ứng dụng mạng chứa một tập hợp các môdun phần mềm được phân tán một cách vật lý trên một hệ mạng diện rộng hỗn tạp. Mỗi môdun phần mềm có thể được thi hành và duy trì bởi những cá thể khác nhau trên những nút mạng. Để thực hiện một ứng dụng mạng thì phải tập hợp một số modun trên

tới một nút mạng đơn. Khả năng liên thao tác để mở một ứng dụng mạng cần sự hỗ trợ của ba hệ thông cơ sở:

1. Các giao diện chuẩn định nghĩa tốt để tích hợp các môđun phần mềm,
2. Năng lực thực hiện môđun phần mềm trên máy tính bất kỳ,
3. Hạ tầng cho cộng tác và vận chuyển modun phần mềm

Để thuận tiện tích hợp phần mềm, Java thông qua mô hình hướng đối tượng, một kiểu lập trình đã được dùng rộng rãi khi phát triển những phần mềm lớn. Ngôn ngữ Java tương tự ngôn ngữ hướng đối tượng C++. Với sự chấp nhận với một ít kiểu dữ liệu, như số và logic, mọi thực thể phần mềm được mô hình hóa như một đối tượng Java. Một đối tượng là một tóm lược của dữ liệu và các thủ tục (hoặc phương pháp) liên quan trên đối tượng đó. Đối tượng được tạo ra bằng việc thuyết minh lớp qua ví dụ. Lớp là một mẫu xác định các biến cũng như những phương pháp chung cho tất cả các đối tượng cùng kiểu (lớp). Lớp này thường chứa đựng lớp khác (thừa kế). Chúng là cơ sở để xây dựng các khối trong chương trình Java. Các file lớp thường dùng phân loại và sắp xếp trong các thư viện lớp được gọi là gói. các gói có thể được nạp cục bộ hay từ xa để khởi tạo đối tượng. Phát triển phần mềm mạng trở thành dễ điều khiển hơn do các thư viện lớp được chia xẻ.

Tiếp cận đặt ra với Java là cho phép chạy mọi modul phần mềm tại mọi nơi theo ngữ nghĩa của khái niệm máy ảo. Hệ thống Java với trình biên dịch và trình phiên dịch. Đầu tiên, chương trình Java được biên dịch thành file lớp chứa các mã trung gian được gọi là applet (tiểu dụng). Tiểu dụng là chương trình độc lập máy và có thể được thông dịch trên mọi máy tính có trình thông dịch Java. Thông dịch trên mã trung gian là kém hiệu quả hơn so với chạy mã máy biên dịch. Tuy nhiên, ưu điểm lớn của cách thức này là mã trung gian được chuyển đi như những TĐ tới bất cứ môi trường nào và chạy trực tiếp không cần dịch lại. Một ứng dụng mạng có thể mang bất cứ một file mã byte nào trên đường truyền để thực hiện. Do các bản sao của file mã không cần lưu cục bộ, bài toán duy trì tính nhất quán cập nhật phiên bản trong phát triển phần mềm cộng tác được loại bỏ.

Java được ràng buộc cẩn thận nhằm đảm bảo tính độc lập máy. Một vài đặc trưng của ngôn ngữ thông dụng là nguyên nhân làm cho các vấn đề liên thao tác hoặc an toàn được loại bỏ khỏi ngôn ngữ. Ví dụ, Java không cung cấp con trỏ, kiểu cấu trúc, chuyển đổi kiểu ngầm định hoặc thừa kế bội. Khái niệm về file đầu (.h) trong C cũng bị loại trừ khỏi Java. Hơn nữa, mọi phương thức và biến trong file lớp Java là được chỉ dẫn bằng tên và được giải quyết trước khi thực hiện. Việc làm chậm giải pháp tên đòi hỏi sự hỗ trợ của dịch vụ tên. Nó cung cấp sự trong suốt truy nhập, trong suốt định vị và an toàn bổ sung.

Hạ tầng để chuyển vận tiểu dụng Java được sáng tỏ tốt nhất nhờ việc tích hợp Java cùng với hệ thống duyệt WWW. Theo nhiều khía cạnh, triết lý của Java cũng rất giống với duyệt Web là sử dụng giao thức giao vận như giao thức chuyển siêu văn bản HTTP để chuyển các modun HTML dọc theo các nút mạng hỗn tạp. HTML là ngôn ngữ đánh dấu độc lập máy để mô tả dữ liệu siêu văn bản. Giống như file lớp trong Java, file HTML là đối tượng có thể chứa các file HTML khác và có thể định vị và liên kết khi dùng bộ định vị tài nguyên tổng thể toàn mạng URL. Tiểu dụng Java có thể hợp nhất trong một file HTML và được thông dịch bởi trình thông dịch Java đã được dựng nội trong trình duyệt. Theo cách đó, trình duyệt vừa có thể hiển thị nội dung dữ liệu siêu văn bản tĩnh vừa có thể chạy linh hoạt tiểu dụng Java. Trình ứng dụng là vô kể. Với trình duyệt đa luồng và Java đa luồng, thì trình duyệt có thể hiển thị đồng thời văn bản cũng như hình ảnh động và trở thành tương tác giữa khách và phục vụ của ứng dụng.

Một cách hiệu quả, trang Web được trình bày dưới một file HTML trở thành lối vào của tiểu dụng Java. Khái niệm thực hiện thông dịch trực tuyến trong Java không phải là mới. Ví dụ, Postscript và dữ liệu đồ họa GIF cũng được thông dịch trong hệ thống trình duyệt. Tuy nhiên, Java là ngôn ngữ đa năng đã được suy nghĩ cẩn thận cho lập trình mạng.

Lưu ý cuối cùng là vấn đề an toàn khi thiết kế Java. An toàn là vấn đề khó tính trong lập trình mạng hệ thống mở. Thêm nữa, để định nghĩa ngôn ngữ chặt chẽ nhằm đề phòng sự lạm dụng của ngôn ngữ, Java là ngôn ngữ định kiểu mạnh giống như Orca. Mọi đối tượng trong Java phải được định kiểu tường minh. Trình biên dịch làm hiệu lực những kiểm tra kiểu tĩnh. Do mỗi máy thấy được tiểu dụng từ bên ngoài theo mã trung gian, cần phải xác minh mã trung gian không phải bị làm giả hay biến dạng. Kiểu và những thông tin điều khiển khác được tích hợp với mọi tiểu dụng. Trước khi thực hiện một tiểu dụng, mã của nó buộc phải được kiểm tra chặt chẽ bộ kiểm tra Java (Java Virifier) xem sự vi phạm về truyền tham số, chuyển đổi kiểu bất hợp pháp, khả năng tràn (vượt trần) và hụt (xuống quá đáy) stack, vi phạm truy nhập và sinh mã trung gian giả bởi trình biên dịch đáng ngờ. Việc kiểm tra lỗi thời gian chạy ở mức tối thiểu nhất nhằm có được sự thực hiện hiệu quả.

Một vần đề về an toàn khác đáng chú ý tới lập trình trên mạng là sự nhái lại đối tượng. Khi tiểu dụng thực hiện có thể gọi một đối tượng khác. File lớp đã được tải cho đối tượng có thể là tiểu dụng đích thực với cùng tên và xuất hiện nhưng có thể không phải từ địa hạt mong muốn. Ví dụ, đối tượng lớp đối với hệ thống file và vào ra I/O nên đến là địa phương. Mỗi lớp file lớp được tương ứng một địa hạt bảo vệ. Địa hạt được phân ra ít nhất là ba mức: máy tính cục bộ, mạng cục bộ và mạng toàn cục mà mức máy tính cục bộ có mức bảo vệ cao nhất. Khi tải một file lớp, các lớp với độ bảo vệ cao hơn được ưu tiên hơn các lớp độ bảo vệ thấp hơn. Hơn nữa, lớp trong một dịa hạt chỉ truy nhập được các phương pháp trong cùng địa hạt. Các phương pháp thuộc các lớp trong một địa hạt khác được truy nhập chỉ khi chúng được khai báo là công cộng. Quy tắc tải các lớp tuân theo Bộ tải lớp Java (the Java Class Loader) do người lập trình Java định nghĩa.

CÂU HỎI VÀ BÀI TẬP

3.1. Khái niệm QT và luồng, ý nghĩa của khái niệm luồng. Đặc điểm chính trong mô hình Client/Server trong hệ phân tán.

3.2. Vai trò của dịch vụ thời gian trong hệ phân tán. Giải pháp đồng hồ vật lý và đồng hồ lôgic trong hệ phân tán.

3.3. Đồng bộ hóa sử dụng biến chung

3.4. Đồng bộ hóa chuyển thông điệp

CHƯƠNG IV. TRUYỀN THÔNG VÀ CỘNG TÁC LIÊN QUÁ TRÌNH

Các QT cộng tác trong hệ thống máy tính tương tác lẫn nhau theo mô hình TTLQT nhằm phối hợp thực hiện. TTLQT và cộng tác QT phân tán là chủ đề chính của chương này. Chương ba đã nhấn mạnh tầm quan trọng của mô hình clien/server đối với truyền thông và quan hệ gắn kết giữa TTLQT và đồng bộ. TTLQT đóng vai trò đáng kể hơn trong hệ phân tán do chỉ có *phương pháp trao đổi dữ liệu QT là CTĐ*. Vì vậy mọi mô hình truyền thông liên QT mức cao đều được xây dựng trên nền CTĐ. Mọi cộng tác QT phân tán đều dựa vào truyền thông liên QT CTĐ.

TTLQT phụ thuộc vào năng lực định vị thực thể truyền thông. Đây chính là vai trò của *dịch vụ tên* trong hệ phân tán. Chương này trình bày ba mô hình truyền thông CTĐ cơ sở và mô hình dịch vụ tên. Tiếp theo là một minh hoạ cộng tác QT phân tán sử dụng hai bài toán kinh điển của TTCTĐ: loại trừ ràng buộc phân tán và chọn thủ lĩnh.

TTLQT có thể được xem xét tại các mức trừu tượng khác nhau. Bảng 4.1 cho năm mức từ mạng tới hệ giao vận và tới các QT ứng dụng. Theo phương diện HĐH phân tán, đầu tiên quan tâm tới ba mức trên chuyển vận TĐ trong các QT phân tán. Chúng là CTĐ, mô hình truyền thông định hướng dịch vụ mức cao sử dụng truyền thông hỏi/đáp và truyền thông giao dịch dựa trên mô hình hỏi/đáp và CTĐ.

Bảng 4.1. cho thấy CTĐ là mức thấp nhất của TT giữa các QT TT. TT hỏi/đáp dựa trên khái niệm client/server. Khi được thi hành như lời gọi thủ tục trong chương trình phân tán, mô hình TT được quy tới lời gọi thủ tục từ xa (RPC). Một cách tự nhiên, hỏi/đáp hoặc RPC dựa trên phương tiện CTĐ cơ sở.

Giao dịch là một dãy các TT hỏi/đáp đòi hỏi TT nguyên tử. Giao dịch biểu diễn đơn vị cơ sở của TT đối với các ứng dụng mức cao, chẳng hạn hệ CSDL. Thực hiện đồng thời các giao dịch cần được đồng bộ để duy trì tính nhất quán của hệ thống. Ngoài ra, khái niệm bộ nhớ chia xẻ lôgic hoặc đối tượng dữ liệu là phương pháp TT khác biệt đáng kể so với ba mô hình CTĐ. Trong hệ thống chỉ với bộ nhớ vật lý phân tán, bộ nhớ chia xẻ được mô phỏng bởi CTĐ. Lợi thế của bộ nhớ chia xẻ lôgic là dễ dàng lập trình, do TT là trong suốt. Giao dịch và bộ nhớ chia xẻ phân tán được trình bày trong các chương 6 và 7.

Bảng 4.1. Các mức khác nhau của TT

TTLQT	Giao dịch
	Hỏi / Đáp (RPC)
	CTĐ
HĐH mạng	Kết nối giao vận
Mạng truyền thông	Chuyển gói

4.1 Truyền thông CTĐ

TĐ là một tập các đối tượng dữ liệu, mà cấu trúc và sự giải thich chúng được xác định bởi các QT ngang hàng với nó. Đối tượng dữ liệu trong TĐ thường được định kiểu nhằm dễ dàng chuyển đổi đối tượng dữ liệu trong hệ thống hỗn tạp. TĐ bao gồm đầu TĐ (chứa thông tin điều khiển phụ thuộc hệ thống) và thân TĐ với kích thước cố định hoặc biến thiên. Trong hệ thống CTĐ, QT TT chuyển các TĐ được đóng gói tới *dịch vụ giao vận hệ thống* cung cấp kết nối truyền TĐ trong mạng. Giao diện tới dịch vụ giao vận là dịch vụ nguyên thủy hiển, chẳng hạn gửi và nhận, hoặc biến thể nào đó của cả hai. Ngữ nghĩa của các dịch vụ nguyên thủy TT này cần xác định hoàn toàn. Các bài

toán chính được đưa ra trong các đoạn sau đây bao gồm TT là trực tiếp hay gián tiếp, kết khối hay không kết khối, tin cậy hay không tin cậy, dùng vùng đệm hay không.

4.1.1. Dịch vụ TT nguyên thủy cơ sở

Hai dịch vụ TT nguyên thủy cơ sở dưới đây là ví dụ để gửi và nhận TĐ. Sẽ là hiệu quả đối với QT ứng dụng khi chỉ rõ thực thể TT và TĐ được truyền:

send (đích, TĐ)

receive (nguồn, TĐ)

trong đó nguồn hoặc đích = (tên QT, liên kết, hộp thư hoặc cổng).

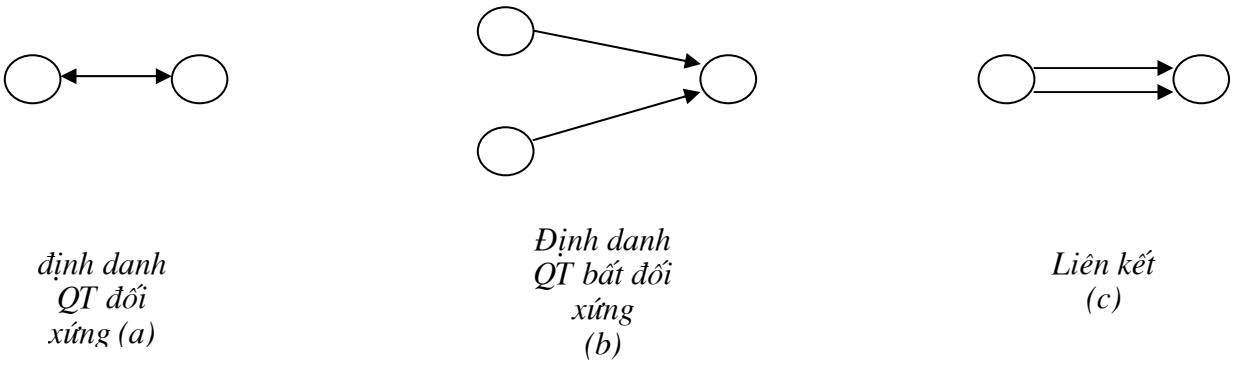
Một câu hỏi nảy sinh trực tiếp từ dịch vụ nguyên thủy là làm thế nào để địa chỉ hóa thực thể TT, nguồn hoặc đích? Dưới đây bàn luận về bốn lựa chọn trên: tên QT, kết nối, hộp thư, cổng.

Đầu tiên, giả sử địa chỉ hóa thực thể TT bằng *tên QT* (tức là định danh QT toàn cục). Khi thi hành thực sự, định danh QT toàn cục có thể được tạo duy nhất qua kết hợp địa chỉ máy chủ mạng với số hiệu QT cục bộ được sinh. Sơ đồ này ngầm định rằng chỉ có một đường TT lôgic trực tiếp tồn tại giữa cặp hai QT gửi và nhận như hình 4.1.a đã chỉ ra. Điều này tương tự TT input/output dùng trong CSP mà đoạn 3.5.3 đã chỉ ra hạn chế của cách tiếp cận này. Sơ đồ địa chỉ được chỉ dẫn là địa chỉ đối xứng do các QT gửi/nhận tương ứng biết rõ nhau trong dịch vụ TT nguyên thủy. Trong một số trường hợp, thuận lợi hơn cho QT nhận là nhận được TĐ từ nguồn chưa biết. Trong trường hợp như thế, *địa chỉ nguồn của DV nguyên thủy nhận* là ***một biến vào*** mà được cho giá trị định danh QT gửi TĐ đó (nếu có một QT nhận). Địa chỉ gửi và nhận là bất đối xứng do chỉ QT gửi cần định vị người nhận. Hình 4.1.b. chỉ ra các trường hợp tổng quát hơn của DV nguyên thuỷ nhận.

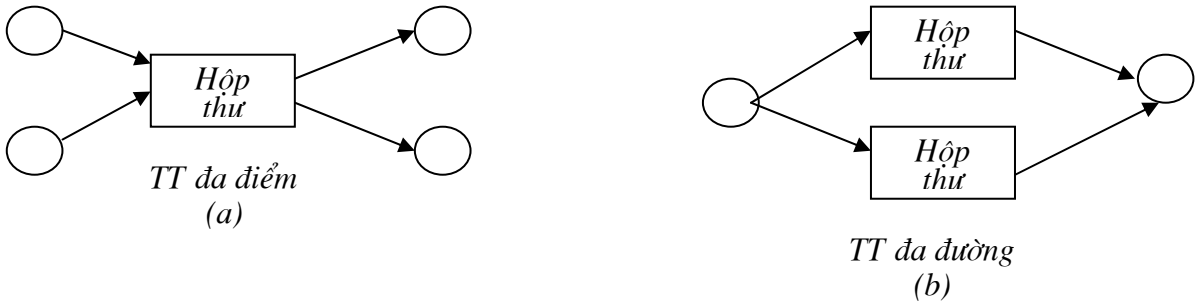
Sơ đồ trên giả thiết tồn tại đường TT trực tiếp giữa cặp hai QT. Thực tế, đường TT là trong suốt hoàn toàn vì vậy đã không chú ý tới kết nối khi giao vận TĐ. Về quan niệm thì đơn giản nhưng để hợp lý chỉ có một đường TT định hướng kép giữa mỗi cặp hai QT TT. Để cho phép đường truyền dữ liệu phức giữa các QT và TT trực tiếp, bắt buộc định danh được mỗi đường đi trong dịch vụ TT nguyên thuỷ. Đòi hỏi này đưa đến khái niệm ***kết nối*** ***hay*** *liên kết*, tương tự với khái niệm chu trình ảo trong mạng TT. TĐ có thể được gửi theo các chu trình ảo khác nhau. Như vậy, điểm TT phức trong một QT cần phải đinh danh bằng việc sử dụng các kết nối khác nhau, mỗi kết nối đó ánh xạ tới một đường TT thực sự. Giống như chu trình ảo, kết nối được tạo và loại bỏ theo yêu cầu. Chúng được nhân hệ thống quản lý cục bộ và là những kênh TT không định hướng. TĐ được gửi qua một kết nối được hướng vào một đường TT mạng và được phân phối tới các máy chủ ở xa. Máy chủ từ xa ánh xạ TĐ tới kết nối đầu vào trong QT nhận. Hình 4.1.c chỉ ra tính hợp lý của việc duy trì hai kết nối giữa các QT khi dùng hai số hiệu kết nối khác nhau. QT đọc cần chú ý kết nối là tương tự với tên điểm vào thủ tục trong cuộc hẹn (đoạn 3.5.3) với lý do là chúng đều cung cấp điểm TT phức trong một QT. Tuy nhiên, giao vận dữ liệu bằng truyền tham số trong cuộc hẹn là định hướng kép.

Dùng tên QT và số hiệu kết nối để định vị các điểm TT cung cấp cơ chế *TT trực tiếp* giữa các QT ngang hàng. Tuy nhiên, đôi khi *TT gián tiếp* cũng được ưa chuộng. QT gửi không quan tâm tới định danh riêng biệt của QT nhận cho đến khi có một QT nhận được TĐ. Tương tự, QT nhận chỉ quan tâm đến chính TĐ mà không cần biết QT gửi. Ví dụ, client phức có thể đòi hỏi dịch vụ từ một trong nhiều dịch vụ phức (định danh của khách có thể được chứa trong chính TĐ). Kịch bản TT này là cồng kềnh khi dùng TT trực tiếp thi hành. Đây là tình huống chung trong cuộc sống hàng ngày, và được

giải quyết bằng hộp thư chung. CTĐ dùng hộp thư chung là sơ đồ TT gián tiếp cung cấp cả TT đa điểm và đa đường một cách hợp lý. Kịch bản này được minh hoạ trong hình 4.2.



Hình 4.1. Dịch vụ TT nguyên thuỷ gửi / nhận trực tiếp



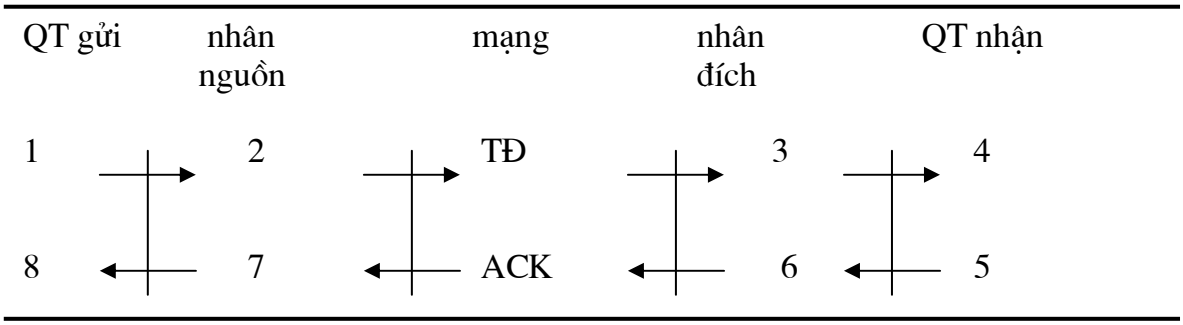
Hình 4.2. Dịch vụ TT nguyên thuỷ gửi / nhận gián tiếp

Về quan niệm, hộp thư là cấu trúc dữ liệu toàn cục chia xẻ của QT sản xuất (gửi) và QT khách hàng (nhận). Dùng hộp thư đòi hỏi sự đồng bộ chính xác dọc theo mạng mà đây là một bài toán khó. Do hộp thư là dùng cho TT, có thể gắn với nó một cấu trúc chuyển vận yếu và thi hành chúng bằng cách dùng vùng đệm và liên kết TT. Cổng là một ví dụ tốt cho hộp thư. Cổng là một khái niệm trừu tượng về một dòng xếp hàng có kích thước cố định hoạt động theo FIFO được nhân duy trì. TĐ có thể gắn vào đuôi và loại bỏ từ dòng đợi bởi các thao tác gửi và nhận xuyên qua một đường TT. Như vậy, cổng tương tự như danh sách ngoại trừ chúng là định hướng kép và có vùng đệm. Các QT TT qua cổng là gián tiếp. Cổng được tạo bởi QT người dùng nhờ lời gọi hệ thống đặc biệt và có thể được phù hợp với QT chủ và đủ năng lực. Chúng được chỉ dẫn bằng số hiệu cổng, mà không thể bị nhầm lẫn với địa chỉ cổng giao vận trong giao vận gói (địa chỉ cổng giao vận là cổng mạng và trong suốt với QT người dùng). Khi thi hành, cổng QT được ánh xạ tới cổng giao vận và ngược lại. Cổng hoặc hộp thư được hình dung như là phục vụ TT và đồng bộ, đã được biện luận trong đoạn 3.6. Thuật ngữ cổng và hộp thư thường được tráo đổi (thay thế nhau) trong một vài tài liệu. Tương tự như socket và cổng trong HĐH UNIX. Socket là giao diện mức cao sử dụng khái niệm cổng. Cổng có chủ nhân là QT riêng biệt. Cổng cung cấp TT nhiều-một (n-1). Hộp thư là đối tượng chia xẻ và cho phép truyền thông nhiều-nhiều (n-n).

4.1.2. Đồng bộ hóa TĐ và vùng đệm

TT CTĐ phụ thuộc một số điểm đồng bộ. Khi gửi TĐ tới đích xa, TĐ đó được chuyển tới nhân hệ thống gửi để thực hiện chuyển giao TĐ cho mạng TT. Cuối cùng, TĐ đi tới được nhân hệ thống đích (ở xa) thực hiện việc trao trả TĐ cho QT đích. Đồng bộ hóa

truyền TĐ xảy xa giữa QT người dùng và nhân hệ thống, nhân và nhân, và QT nguồn và QT đích. Hình 4.3. chỉ rõ các giai đoạn khác nhau của CTĐ trong hệ thống.



Hình 4.3. Các giai đoạn đồng bộ TĐ

Dịch vụ nguyên thủy gửi và nhận được coi là kết khối nếu QT gọi cần kết khối để phân phối hay nhận TĐ tương ứng. Hầu hết hệ thống cho phép chọn dịch vụ nguyên thủy gửi/nhận kết khối hoặc không kết khối. Hầu hết ngầm định gửi không kết khối và nhận kết khối. Lý do là để thuận tiện, giả thiết rằng phân phối TĐ là đáng tin cậy và QT gửi có thể tiếp tục công việc một cách hiệu quả sau khi TĐ đã được dàn xếp và nhân bản tới nhân gửi. Mặt khác, QT nhận cần chờ cho đến khi TĐ xuất hiện để thực hiện công việc của mình. Tuy nhiên, không phải mọi trường đều như vậy. Chẳng hạn, QT gửi có thể mong muốn đồng bộ với QT nhận hoặc QT nhận mong muốn TĐ từ QT gửi phức và không thể không đủ chỗ cho thao tác nhận riêng biệt. Tại phía nhận, kết khối là hoàn toàn rõ ràng; nó cần được kết khối theo sự xuất hiện của TĐ. Về phía QT gửi, rắc rối hơn đôi chút. QT gửi nên chờ việc nhận được TĐ của nhân nguồn, nhân đích, hoặc QT đích hoặc thậm chí hoàn thiện một số thao tác của QT nhận? Danh sách dưới đây chỉ dẫn năm chức năng khác nhau của dịch vụ nguyên thủy gửi theo sơ đồ ở hình 4.3:

1. *Gửi không kết khối*, 1+8: QT gửi được loại bỏ sau khi TĐ đã được dàn xếp và sao tới nhân nguồn.
2. *Gửi kết khối*, 1+2+7+8: QT gửi được loại bỏ sau khi TĐ đã được truyền tới mạng
3. *Gửi kết khối tin cậy*, 1+2+3+6+7+8: QT gửi bị loại bỏ sau khi TĐ đã được nhân đích nhận xong.
4. *Gửi kết khối tường minh*, 1+2+3+4+5+6+7+8: QT gửi bị loại bỏ sau khi TĐ đã được QT nhận xong
5. *Hỏi và đáp*, 1-4, dịch vụ, 5-8: QT gửi bị loại bỏ sau khi TĐ đã được xử lý bởi QT nhận và lời đáp trở lại QT gửi.

Phương án đầu tiên là *gửi dị bộ* còn những phương án khác đều là *gửi đồng bộ*. Phương án cuối cùng chính là TT clien/server. Trong gửi dị bộ, QT gửi bị kết khối nếu nhân tại nó chưa sẵn sàng tiếp nhận TĐ, có thể do thiếu không gian vùng đệm. Đây là đòi hỏi tối thiểu nhất vì rất nguy hiểm nếu QT gửi tiếp tục công việc (chẳng hạn, tạo ra một TĐ mới) trước khi nhân gửi nắm điều khiển TĐ. Khi giả thiết là gửi/nhận dị bộ, ta mong muốn rằng dịch vụ nguyên thủy cần cho một mã quay về cho biết kết quả thành công hay thất bại của thao tác để qua phân tích mã quay về để hoặc gửi TĐ tiếp theo hoặc xử lý lỗi.

Trong sơ đồ hình 4.3, ngầm định tồn tại vùng đệm trong nhân gửi, nhân nhận và mạng TT. Vùng đệm trong nhân hệ thống cho phép TĐ được gửi đến thậm chí khi TĐ trước nó chưa được phân phối. Do QT gửi và nhận chạy dị bộ, chúng tạo ra và xử lý các TĐ theo các mức độ (tốc độ) khác nhau. Do có vùng đệm, sự không đồng nhất này trở nên êm ả. Thêm nữa, khả năng QT gửi bị kết khối được rút gọn và thông lượng truyền tổng

thể TĐ được tăng lên. Vùng đệm được dùng để điều khiển lưu lượng trong mạng TT. Trong HĐH, thông thường vùng đệm được chia xẻ bởi TT gửi và nhận đa thanh phần. Quản lý vùng đệm hiệu quả trở thành một bài toán quan trọng. Quản lý vùng đệm không chính quy có thể trở thành nguyên nhân bế tắc TT.

Về lôgic, có thể kết hợp vùng đệm trong nhân gửi, nhân nhận, và mạng thành một vùng đệm lớn. QT gửi tạo ra TĐ và chèn chúng vào vùng đệm còn QT nhận xóa khỏi vùng đệm và sử dụng chúng. Nếu vùng đệm là không giới hạn, QT gửi dị bộ là không kết khối. Một trường hợp đặc biệt khác là mọi thành phần là vắng vùng đệm (zero-buffer). Trong trường hợp này, QT gửi và QT nhận bắt buộc phải đồng bộ (trách nhiệm đồng bộ hóa dành cho người viết chương trình các QT này) để đủ năng lực truyền TĐ (bất cứ TĐ nào xuất hiện thì trước hết phải đợi TĐ trước đó). Điều này tương tự như khái niệm cuộc hẹn và là một kiểu gửi/nhận kết khối tường minh.

4.1.3. API ống dẫn và Socket

Như đã nói ở trên, tồn tại lượng lớn và đa dạng các dịch vụ nguyên thủy TT CTĐ với các khái niệm và giả thiết khác nhau. Khi TT được thực hiện nhờ một tập hoàn toàn xác định các giao diện chương trình ứng dụng chuẩn (API) sẽ tạo thuận lợi cho người dùng và hiệu quả cho hệ thống. TT QT người dùng sử dụng một API độc lập với môi trường TT hạ tầng. Ống dẫn (pipe) và socket là hai API TTLQT được sử dụng rộng rãi trong cả hai môi trường UNIX và Windows.

Như trình bày trong đoạn 3.5.3 thì chia xẻ kênh TT về mặt lôgic là tương đương với chia xẻ biến. Cả hai đều là chia xẻ đối tượng. Trong thực tế, kênh TT được thi hành bởi chia xẻ lưu trữ, chẳng hạn không gian nhân, bộ nhớ, hoặc file. Trong hệ đơn xử lý hỗ trợ QT TT có thể mô phỏng kênh TT nhờ chia xẻ bộ nhớ trong không gian nhân. QT người dùng thấy được kênh TT theo trình diễn bởi API. Chi tiết nội tại và thi hành, chẳng hạn như dung tích của kênh và đồng bộ truy nhập bộ nhớ, được nhân quản lý và trong suốt với người dùng. Ống dẫn được thi hành bằng một vùng đệm dòng byte FIFO kích thước cố định được nhân duy trì. Được hai QT TT sử dụng, phục vụ ống dẫn như một kết nối TT không định hướng mà một QT có thể ghi dữ liệu vào đuôi của ống dẫn và một QT khác có thể đọc từ đầu của nó. Ống dẫn được khởi tạo bởi lời gọi hệ thống ***pipe*** cho hai đặc tả ống dẫn (tương tự như đặc tả file), một để đọc và một để ghi. Kịch bản điển hình để ống dẫn giữa hai QT là vì một QT phải khởi tạo ống dẫn, fork QT khác, gắn QT cha vào đầu đọc ống dẫn và gắn đầu ghi ống dẫn tới QT con. Như vậy một dòng dữ liệu một chiều trở thành chuyển dịch giữa QT cha và con khi sử dụng các thao tác ghi và đọc bình thường. Đặc tả ống dẫn được các QT TT chia xẻ. Điều này ngụ ý rằng ống dẫn được sử dụng chỉ với các QT có quan hệ với nhau (tức là, QT được khởi tạo thông qua thao tác fork). Trong điều kiện thông thường, QT đọc và ghi được giả thiết là chạy đồng thời đối với mọi ống dẫn được tạo. Ống dẫn chỉ tồn tại trong khoảng thời gian cả hai QT đọc và ghi hoạt động. Thao tác ghi ống dẫn không kèm thao tác đọc tương ứng là vô nghĩa do ống dẫn ngừng tồn tại khi QT ghi kết thúc.

Dữ liệu trong ống dẫn mặc nhiên là dòng byte liên tục. Tiếp cận này được chọn nhằm khớp với giả thiết chung cấu trúc file hướng byte của UNIX. Đôi khi mong muốn rằng là dòng dữ liệu cấu trúc, chẳng hạn TĐ độ dài biến đổi trong kênh và khái niệm ống dẫn có thể được mở rộng để bao gói cả TĐ. Kiểu kênh TT này được hiểu là *dòng xếp hàng TĐ*. Dòng xếp hàng TĐ được thi hành trong không gian bộ nhớ của nhân. Nhiều hệ thống cung cấp dòng xếp hàng TĐ như là một IPC API.

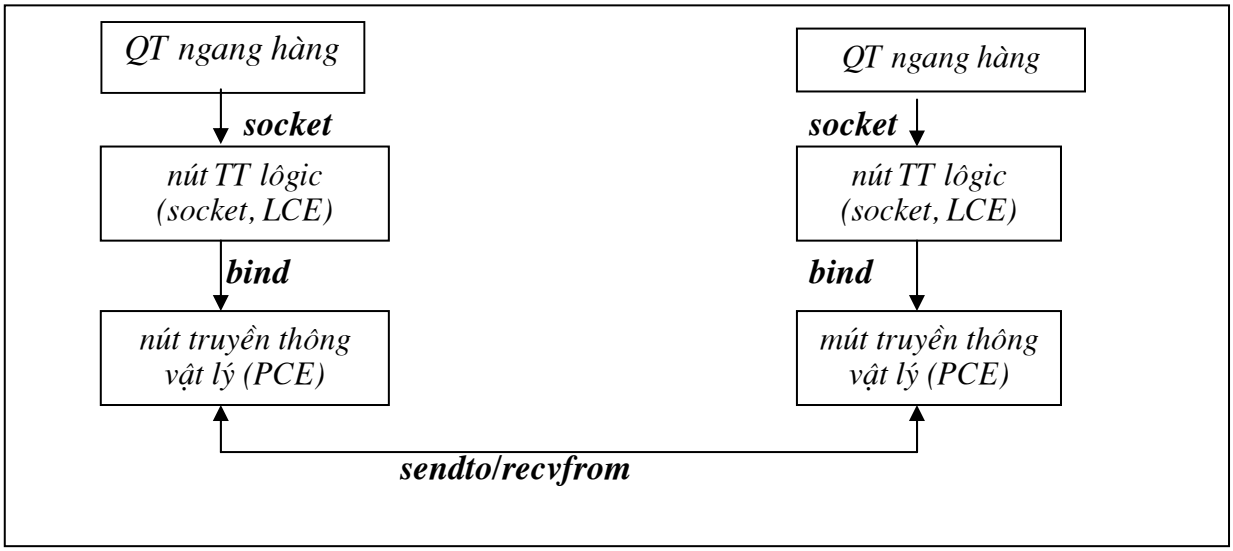
Với những QT không quan hệ (fork), cần định danh ống dẫn vì đặc tả ống dẫn không thể chia xẻ. Một giải pháp là thay cấu trúc dữ liệu ống dẫn nhân bằng một *file FIFO đặc biệt*. File FIFO đặc biệt được định danh duy nhất bằng ***tên đường*** tương tự như file

thông thường. Ống dẫn với tên đường được gọi là ***ống dẫn có tên***. Với một tên duy nhất, ống dẫn có tên có thể được chia xẻ giữa các QT rời rạc xuyên qua các máy tính khác nhau với một hệ thống file chung. Do ống dẫn có tên là file thì các QT TT không cần đồng thời tồn tại. QT ghi có thể ghi xong dữ liệu tới một ống dẫn có tên và kết thúc trước khi một thao tác đọc file xuất hiện. Ống dẫn có tên dùng ngữ nghĩa của một file thông thường. Chúng được khởi tạo bởi câu lệnh ***open*** trước khi tạo ra truy nhập tới file FIFO.

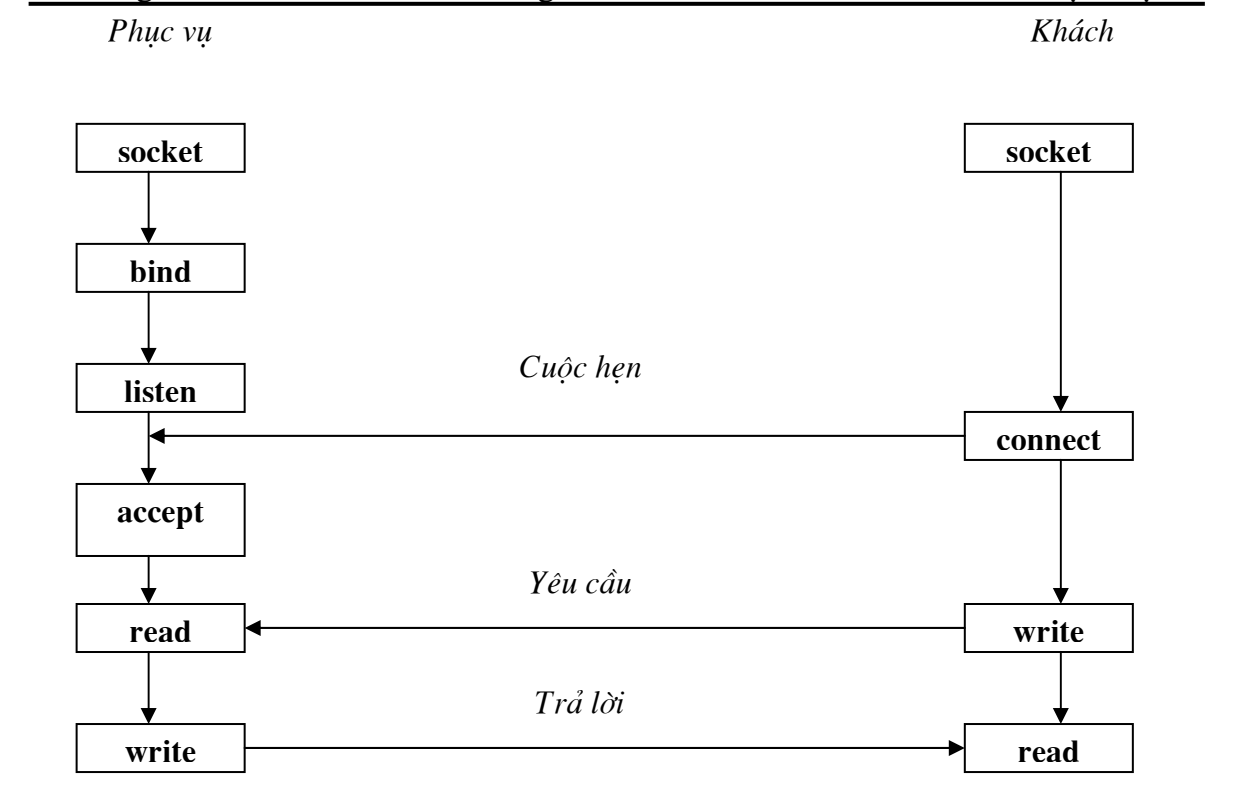
Ống dẫn và ống dẫn có tên thi hành bài toán IPC giữa ***nhà sản xuất*** và ***khách hàng***. Trong bài toán nhà sản xuất và khách hàng, QT sản xuất (gửi) và QT khách (nhận) tương tác nhau thông qua một vùng đệm chung để hoàn thành TTLQT. Vấn đề đồng bộ là loại trừ ràng buộc đối với truy nhập vùng đệm và cộng tác có điều kiện khi vùng đệm là đầy hoặc rỗng. Truy nhập vùng đệm được chú ý như khoảng tới hạn mà cần được giám sát. Điều kiện tràn hoặc rỗng của vùng đệm là tương tự kết khối của gửi (sản xuất) và nhận (khách hàng) với một vùng đệm cố định. Thi hành ống dẫn và ống dẫn có tên đơn thuần bảo đảm tính nguyên tử của vùng đệm nhân chia xẻ và file FIFO đặc biệt và việc kết khối thao tác ghi và đọc khi vùng lưu trữ chia xẻ là đầy hoặc rỗng. Các byte được ghi từ QT phức tới ống dẫn được đảm bảo không khi nào là chen lẫn. Cẩn thận đặc biệt khi ghi dữ liệu riêng tới ống dẫn trước khi nó trở nên đầy. Hoặc toàn bộ các byte của TĐ được ghi vào ống dẫn hoặc không.

Dùng ống dẫn định danh gặp một hạn chế từ tên miền đơn trong hệ thống file chung. Để đạt được TT QT liên miền mà không có cấu trúc dữ liệu hoặc file có tên duy nhất và được chia xẻ, cần có một IPC API chạy trên đỉnh của dịch vụ giao vận. Hai API TT liên QT liên miền được dùng rộng rãi nhất là socket Berkeley và Giao diện mức giao vận hệ thống 5 (TLI). Socket Berkerley là ví dụ minh họa API TT.

Việc đặt tên kênh TT qua một miền hỗn tạp là không khả thi. Tuy nhiên, kênh TT có thể được hình dung như một cặp gồm hai đầu mút TT. Socket là mút TT của kết nối TT được quản lý bởi dịch vụ giao vận. Tương tự việc sử dụng ống dẫn cho phép file I/O có ngữ nghĩa đối với việc *đọc từ* và *ghi tới* ống dẫn, mô hình I/O mạng socket dựa trên I/O File quy ước. Trừu tượng hóa I/O mạng như I/O file làm tăng tính trong suốt truy nhập trong hệ thống. Socket được tạo ra nhờ lời gọi hệ thống ***socket*** cho một đặc tả socket phục vụ các thao tác I/O mạng tiếp sau, bao gồm cả đọc/ghi hướng file và gửi/nhận đặc trưng TT. Lời gọi hệ thống socket cũng được sử dụng trong nhiều giao thức mạng như TCP, UDP và IP. TCP là giao thức giao vận dòng thực hiện hướng kết nối và UDP là giao thức giao vận sơ đồ không kết nối. Chúng là hai giao thức giao vận chính. IP được dùng để truyền dòng gói dữ liệu và là giao thức tầng mạng không kết nối trong bộ giao thức Internet. Đặc tả socket là nút TT logic (LCE: Logic Communication EndPoint) cục bộ đối với một QT; nó bắt buộc phải phù hợp với nút TT vật lý (PCE: Physic CE) để truyền dữ liệu. Nút TT vật lý được đặc tả bởi địa chỉ máy chủ mạng và cặp cổng giao vận. Địa chỉ máy chủ mạng là toàn cục, trong khi số hiệu của giao vận được sinh cục bộ bởi dịch vụ giao vận. Việc phù hợp một LCE với một PCE được thi hành bằng lời gọi hệ thống ***bind***. Hình 4.4. chỉ ra một ví dụ TT ngang hàng không kết nối dùng các lời gọi hệ thống ***socket, bind*** và ***sendto/recvfrom***. Do TT là không kết nối nên mỗi lời gọi ***sendto/recvfrom*** bắt buộc chứa đặc tả socket cục bộ và PCE từ xa.



Trong TT socket không kết nối mỗi QT ngang hàng bắt buộc phải biết PCE từ xa của nó. Có thể được loại bỏ việc gọi tên hiển của PCE từ xa trong lời gọi gửi/nhận nếu lời gọi ***socket*** kết nối ràng buộc một LCE cục bộ với PCE từ xa của nó trước khi bắt đầu truyền dữ liệu. Sau thao tác kết nối, truyền dữ liệu có thể đơn giản là send/recv hoặc write/read không có đặc tả của PCE từ xa. Lời gọi ***socket*** kết nối thông thường được dành riêng cho TT Client/Server hướng kết nối. Đối với TT Client/Server, dịch vụ cần



Hình 4.5. Truyền thông socket Client/Server hướng kết nối

có được PCE rõ ràng. Một phục vụ sẽ cần TT với khách phức có PCE chưa biết. Khách đưa ra một lời gọi ***connect*** tới phục vụ để hẹn (cuộc hẹn), với yêu cầu khách nhờ một ***accept*** và thiết lập có kết quả một kết nối tới khách đó. Về khái niệm, điều này tương đương với thi hành cuộc hẹn Ada trong TT liên miền. Hình 4.5. minh họa TT socket

Client/Server hướng kết nối. Trong thi hành UNIX, lời gọi socket ***listen*** được dùng để chỉ ra phục vụ sẽ chấp nhận một kết nối và đặc tả độ dài dòng xếp hàng (bao nhiêu lời hỏi xảy ra có thể xếp hàng). Lời gọi ***accept*** hẹn với lời gọi ***connect*** được tích lũy lại trong dòng xếp hàng ***listen***. Một lời gọi ***accept*** sẽ kết khối nếu chưa có một ***connect*** giải quyết. Nếu có, nó xoá bỏ yêu cầu ***connect*** từ dòng đợi và đưa ra một đặc tả socket mới được dùng để TT với khách đã được kết nối. Đặc tả socket cũ còn lại trong dịch vụ cho các yêu cầu khách khác. Trong thi hành phục vụ đồng thời, QT (luồng) con là được phân nhánh đối với mỗi kết nối sử dụng đặc tả socket mới.

4.1.4. Socket an toàn

Socket đã trở thành API CTĐ phổ biến nhất trong cộng đồng Internet. Do việc sử dụng rộng rãi các ứng dụng Windows mà nhóm chuẩn WinSock, bao gồm hơn 30 hãng công nghiệp (kể cả MicroSoft) đã phát triển một socket Windows chuẩn (WinSock). WinSock bắt nguồn từ socket Berkeley. Nó gồm một tập công phu các API và được mở rộng nhằm cung cấp tính trong suốt giao vận hoàn hảo khi sử dụng giao diện cung cấp dịch vụ (SPI: Service Provider Interface) trừu tượng làm dễ dàng tương thích *plug-in* cho hầu hết các giao thức giao vận. Phiên bản gần nhất cũng chứa tầng socket an toàn (SSL: Secure Socket Layer).

Đòi hỏi an toàn TT trên Internet đã thúc đẩy IETF (Internet Engineering Task Force) phát triển SSL. Mục tiêu SSL là cung cấp:

- Bảo mật trong TT socket khi dùng mã đối xứng để mã hoá dữ liệu
- Toàn vẹn dữ liệu trong socket khi kiểm tra tính toàn vẹn TĐ
- Xác thực phục vụ và khách khi dùng mã hóa khóa công khai bất đối xứng.

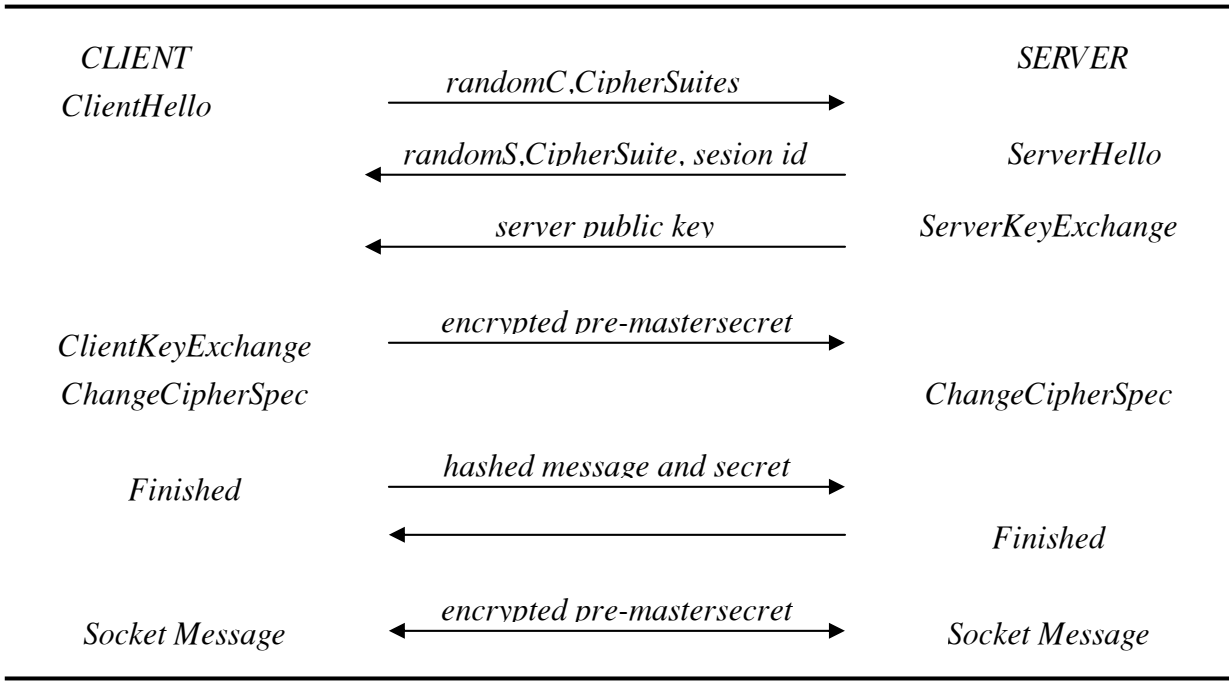
Điểm chủ yếu của SSL chứa trong hai mức giao thức: một giao thức Handshake và một giao thức Record Layer. Giao thức Handshake tương ứng thiết lập các *khóa ghi* (khóa phiên TT để bí mật dữ liệu) và *MAC* (Message Authentication Check để toàn vẹn dữ liệu) bí mật và xác nhận tính xác thực của phục vụ và khách. Giao thức Record Layer thích hợp để phân đoạn, nén/giãn nén và mã hóa/giải mã các bản ghi của TĐ. Kết quả cuối cùng của giao thức Handshake là một cấu trúc dữ liệu chia xẻ (được gọi là mastersecret) chỉ khách và phục vụ biết được, mà có thể được biến đổi thành ***write key*** và một ***MAC secret*** để TT an toàn bằng Record Layer.

Hình 4.6. trình bày một kịch bản đơn giản của giao thức Handshake SSL. Khách muốn liên lạc với phục vụ bằng cách gửi TĐ *ClientHello* tới phục vụ đó. Thành phần chính của TĐ chứa một số ngẫu nhiên (randomC) và một tập thuật toán mật mã (CipherSuites). Số ngẫu nhiên được dùng để tính toán *mastersecret* quyết định. CipherSuites là một danh sách lựa chọn mã hóa được phục vụ đàm phán và chọn. Phục vụ trả lại cho khách một TĐ phục vụ*Hello* chứa một số ngẫu nhiên *randomS*, một thuật toán mã hóa *CipherSuite* được chọn và một định danh phiên cho kết nối.

Tại thời điểm này, phục vụ có thể xác nhận định danh của nó bằng việc gửi một giấy chứng nhận tới khách. Giấy chứng nhận được cho bằng giấy xác thực (CA) nhóm ba. Giấy chứng nhận được QT cấp giấy ký khi dùng khóa bí mật của nó và như vậy không thể dễ giả mạo. SSL dùng xác nhận X.509. Phục vụ có thể yêu cầu giấy chứng nhận của khách. Mỗi một chứng nhận mang thành phần khóa công khai trong cặp gồm khóa công khai và khóa bí mật của đối tượng được ghi nhận (khách hoặc phục vụ). Khách cần khóa công khai của phục vụ để biến đổi thông tin bí mật tới phục vụ. Mã hóa khóa công khai được trình bày trong chương sau. Phương pháp cặp khóa kép (công khai và bí mật) được coi là một thuật toán mã hóa. Với nó, một TĐ được mã hóa bởi một khóa công khai có thể được giải mã bằng khóa bí mật tương ứng và ngược lại. Khóa công

khai được ghi nhận bằng thông tin công khai còn khóa bí mật chỉ có các đối tượng biết. Để đơn giản hóa trong trình bày giao thức Handshake SSL ở hình 4.6 đã bỏ qua việc xác nhận tính hợp lệ của các giấy chứng nhận.

Không cần giấy chứng nhận, một phục vụ nặc danh có thể gửi khoá công khai của nó trong TĐ phục vụ*KeyExchange* tới Khách. Khóa công khai này không cần phải là khóa



Hình 4.6. Giao thức Handshake

đã được ghi nhận. Phục vụ sinh tạm thời khóa công khai để sử dụng theo từng lần yêu cầu của khách. Khách đáp lại bằng một TĐ *ClientKeyExchange* mang một pre-mastersecret mã hóa theo khóa công khai tạm thời của phục vụ. Chỉ có phục vụ với khóa bí mật tương ứng mới giải mã được pre-mastersecret. Lúc đó, cả khách và phục vụ chia xẻ pre-mastersecret và hai số ngẫu nhiên. Cả hai QT độc lập áp dụng hàm băm một chiều tới thông tin chia xẻ để chuyển pre-mastersecret quyết định chứa khóa ghi (write key) và MAC bí mật. Các khóa và MAC bí mật này được dùng để liên kết với bộ mật mã vừa được đàm phán. Chúng được ChangeCipherSpec tạo hiệu quả nhằm thay thế bộ mật mã cũ bằng một bộ mới. Các TĐ ***finished*** chấm dứt việc bắt tay. Chúng cũng được dùng để xác minh việc trao đổi khóa và xác thực có thành công hay không. Việc kiểm tra thông qua xác nhận TĐ ***finished*** chứa kết quả băm của mastersecret được móc nối với mọi TĐ bắt tay.

TT socket an toàn được bắt đầu sau khi TĐ ***finished*** đã được trao đổi và kiểm tra. Mọi TĐ socket tiếp sau được mã hóa theo thuật toán mã hóa và khóa ghi bí mật đã được thiết lập cho đến khi phiên được thương lượng lại. Mọi TĐ chứa một bộ kiểm tra xác thực TĐ là kết quả băm TĐ với MAC bí mật. Không có MAC bí mật, sản xuất MAC cho TĐ tạm thời trở nên bất hợp lý. TĐ socket được xử lý bởi Record Layer trở thành bí mật và bền vững. Khái niệm giao thức socket an toàn vẫn đang được tiếp tục tiến hóa và cải tiến.

4.1.5. Truyền thông nhóm và phân phát bội (multicast)

Mô hình TT CTĐ được trình bày trên đây dùng cho TT điểm-điểm. Mục này mô tả nhu cầu và thi hành TT nhóm đa điểm. Cần lưu ý là nhóm là bản chất để phát triển phần mềm cộng tác trong hệ phân tán hay tự trị. Quản trị nhóm các QT hoặc đối tượng cần có cơ chế TT phân phát bội để gửi TĐ tới các thành viên trong nhóm. Tồn tại hai kịch

bản ứng dụng TT phân phát bội. Đầu tiên là một khách mong muốn cố níu kéo một dịch vụ từ bất kỳ phục vụ nào miễn là có khả năng đáp ứng dịch vụ. Thứ hai là một khách đòi hỏi dịch vụ từ tất cả các thành viên trong nhóm phục vụ.

Trong trường hợp đầu tiên, không cần phải tất cả phục vụ đáp ứng lại mà chỉ cần một phục vụ. Phân phát bội được thực hiện trên cơ sở *cố gắng nhất* (best-effort) và được lặp lại nếu cần thiết. Hệ thống chỉ cần đảm bảo phân phát bội TĐ tới các QT không bị mắc lỗi có thể đạt được. Cách như vậy gọi là phân phát bội *cố gắng nhất*.

Trong trường hợp sau, cần đảm bảo là mọi phục vụ đều nhận được yêu cầu và tính bền vững trong các phục vụ có thể được duy trì. TĐ phân phát bội cần được đáp ứng cho tất cả các phục vụ nhận hoặc không một phục vụ nào (tức là toàn bộ hoặc không cái nào); cách này thường được gọi là *phân phát bội tin cậy*. Đòi hỏi toàn bộ hoặc không cái nào có nghĩa là TĐ phân phát bội nhận được cần được đưa vào vùng đệm trước khi phân phối cho QT ứng dụng. Chú ý trong phân phát bội tin cậy đồng bộ ảo, TĐ có thể được phân phối trước khi nhận được (Đồng bộ ảo được thảo luận ở phần sau).

Ihi hành phân phát bội phức tạp hơn vì gặp nhiều thiếu thốn do chưa có phân phát bội nguyên tử. Lỗi của QT nhận hoặc kết nối truyền thông có thể được QT khởi tạo TĐ phát hiện khi sử dụng cơ chế quá hạn hoặc xác nhận. QT khởi tạo sau đó có thể thoát ra hoặc tiếp tục phân phát bội bằng cách loại bỏ thành viên lỗi trong nhóm. Lỗi của khởi tạo một chiều (haft-way) trong phân phát bội chỉ mới được giải quyết một cách giả định. Rất khó khăn để xác định khởi tạo là có lỗi hay không. Để xác định thoát từ lỗi hoặc toàn bộ các bộ phận của phân phát bội là hoàn thiện, một trong các QT nhận bắt buộc được chọn như một khởi tạo mới. Kỹ thuật thông thường còn đòi hỏi các QT nhận phải đưa vào bộ đệm phân phát bội cho tới khi TĐ đã trở nên an toàn cho phân phối. Lỗi được kiểm soát nhờ hệ thống ảo. Phân phát bội bỏ qua đồng bộ ảo là không thực sự tin cậy; chúng chỉ là *cố-gắng-nhất*.

Quan hệ trực tiếp với bài toán phân phối tin cậy là bài toán về thứ tự phân phối các TĐ. Khi TĐ phức là phân phát bội tới cùng một nhóm, chúng xuất hiện tại các thành viên khác nhau trong nhóm theo các thứ tự khác nhau (do tính biến động của độ trễ trong mạng).

Hình 4.7 cho một số ví dụ TT nhóm yêu cầu thứ tự TĐ: G và s tương ứng biểu diễn nhóm và nguồn TĐ. QT s có thể đứng ngoài nhóm hoặc là một thành viên của nhóm. Giả thiết rằng TĐ phân phát bội cần được nhận và phân phối ngay lập lức theo thứ tự chúng được gửi. Nếu giả thiết này là đúng thì công việc lập trình nhóm đơn giản hơn rất nhiều. Tuy nhiên điều đáng tiếc là giả thiết này không có thực và thiếu ý nghĩa vì trong hệ phân tán không có được thời gian toàn cục và giao vận TĐ trong mạng gặp độ trễ TT đáng kể và không ổn định. Về ngữ nghĩa, phân phát bội có thể được xác định sao cho TĐ được nhận theo thứ tự khác nhau tại các nút khác nhau có thể được sắp xếp lại và phân phối tới QT ứng dụng theo quy tắc chặt chẽ nhỏ hơn. Thứ tự phân phát bội dưới đây được xếp theo độ tăng của tính chặt chẽ:

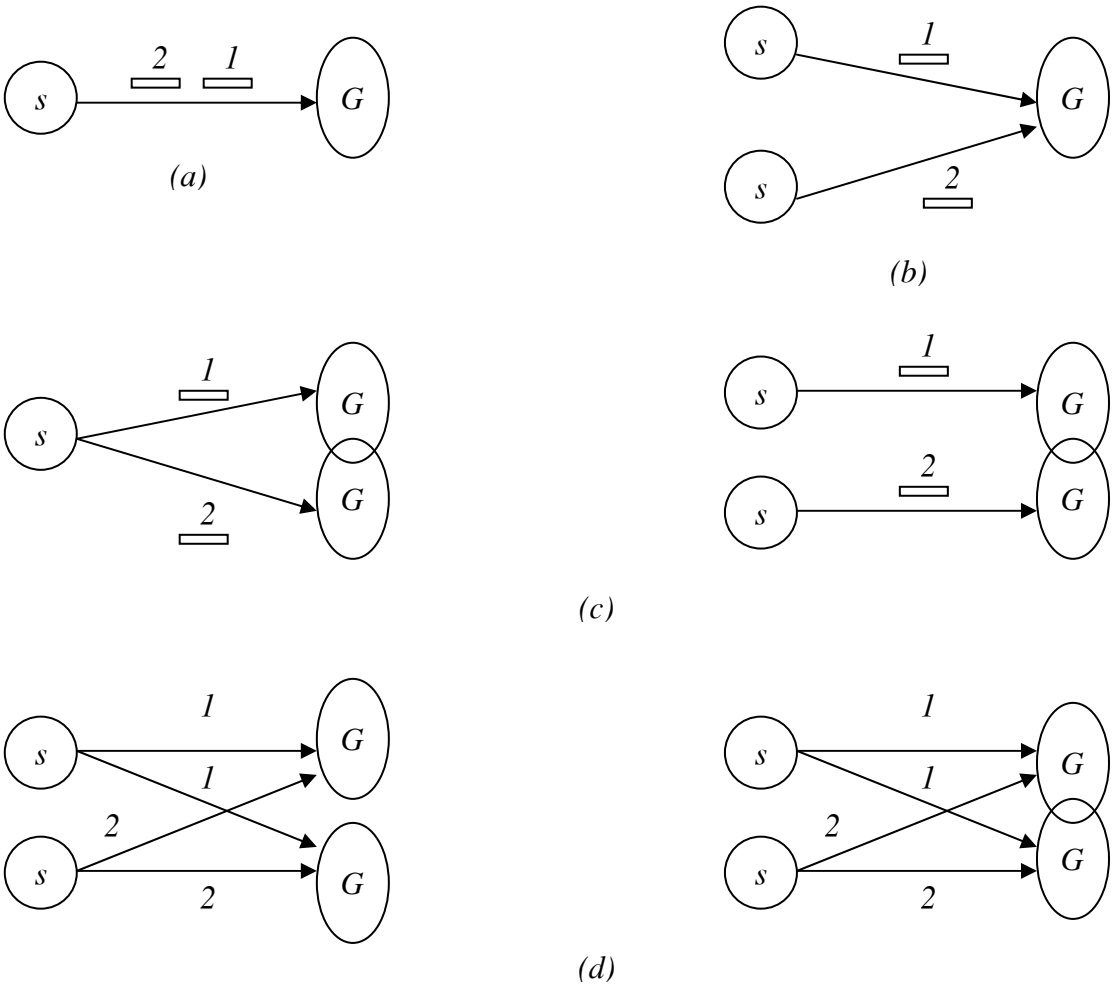
+ Thứ tự FIFO: TĐ phân phát bội từ nguồn đơn được phân phối theo thứ tự chúng được gửi.

+ Thứ tự nhân quả: TĐ quan hệ nhân quả từ nguồn phức được phân phối theo thứ tự nhân quả của chúng.

+ Thứ tự tổng: Mọi TĐ phân phát bội tới một nhóm được phân phối tới mọi thành viên của nhóm theo cùng thứ tự. Một thứ tự tin cậy và tổng được gọi là thứ tự nguyên tử.

Tại mỗi nút, chương trình điều khiển TT chịu trách nhiệm nhận TĐ và sắp xếp lại theo thứ tự tới QT ứng dụng. Điều này tương tự như tính chất mô hình bất biến của hệ thống

file phân tán và hệ thống bộ nhớ chia xẻ phân tán. Chúng là tương tự nhau trong bối cảnh phân tán.



Hình 4.7. Truyền thông nhóm và thứ tự TĐ

Thi hành theo thứ tự FIFO (hình 4.7a) là dễ dàng. Do chỉ có các TĐ được gửi từ cùng một QT khởi tạo, các TĐ này được gán số hiệu TĐ tuần tự. Điều khiển TT có thể làm trễ TĐ hoặc loại bỏ các TĐ lặp khi sử dụng dãy số hiệu tuần tự này. Dãy số hiệu tuần tự TĐ là cục bộ đối với mỗi nguồn TĐ và vì vậy không thể kết hợp các TĐ từ các nguồn khác nhau (xem hình 4.7 b). Thứ tự nhân quả và thứ tự tổng của TĐ phân phát bội từ các nguồn khác nhau là công phu hơn.

Hai TĐ được gọi là có quan hệ nhân quả với nhau nếu một TĐ được sinh ra sau khi đã tiếp nhận xong cái còn lại. Thứ tự TĐ nhân quả cần được trình bày tại mọi nút (phía) do nội dung của TĐ thứ hai có thể được tác động theo kết quả xử lý TĐ đầu tiên. Quan hệ nhân quả này có thể trải dọc qua một vài thành viên trong nhóm do tính bắc cầu của quan hệ nhân quả. Thi hành thứ tự nhân quả các TĐ bằng cách mở rộng số hiệu tuần tự thành vector số hiệu tuần tự, $S=(S_1, S_2, ..., S_n)$ được mỗi thành viên duy trì. Mỗi S_k trình bày số hiệu TĐ sẽ nhận được từ thành viên k của nhóm. Khi thành viên i phân phát bội một TĐ mới m, nó làm tăng S_i lên 1 (dấu hiệu cho biết số lượng TĐ mà i đã phân phát bội) và gắn vector S với m. Khi nhận được TĐ m có vector tuần tự $T=(T_1,$

T_2, ..., T_n) từ thành viên i, thành viên j hoặc tiếp nhận hoặc làm trễ phân phối m theo các luật dưới đây (Chú ý S_i là thành phần vector số hiệu tại thành viên j):

• Tiếp nhận TĐ m nếu $T_i=S_i+1$ và $T_k \le S_k$ với mọi $k \neq i$. Điều kiện đầu tiên ($T_i=S_i+1$) chỉ ra rằng thành viên j mong chờ TĐ tiếp sau theo dãy từ thành viên i. Điều kiện thứ hai xác minh rằng thành viên j đã phân phát mọi TĐ phân phát bội mà thành viên i đã phân phát trước khi nó phân phát bội m (có thể một vài cái nữa). Như vậy, j đã thực sự phân phát mọi TĐ đứng trước (nhân quả) m.

• Làm trễ TĐ m nếu hoặc $T_i>S_i + 1$ hoặc tồn tại một số $k \neq i$ mà $T_k > S_k$. Trường hợp đầu tiên, một vài TĐ phân phát bội trước đây từ thành viện i đã bị thất lạc mà thành viên j đã không nhận được. Trường hợp thứ 2, khi thành viên i phân phát bộ m thì nó đã nhận được nhiều TĐ phân phát bội từ các thành viên khác trong nhóm hơn so với thành viên j. Trong cả hai trường hợp, TĐ bắt buộc phải bị làm chậm để đảm bảo tính nhân quả.

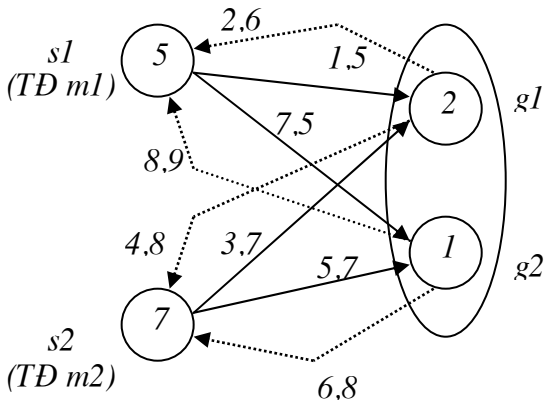
• Loại bỏ TĐ nếu $T_i \le S_i$. Việc sao lặp TĐ từ thành viên i đã được bỏ qua hoặc loại bỏ bởi thành viên j.

Giao thức thứ tự nhân quả này giả thiết rằng phân phát bội trong một nhóm đóng (tức là nguồn của phân phát bội cũng là một thành viên của nhóm) và phân phát bội không thể mở rộng dọc theo nhóm (mục sau sẽ bàn luận về việc này).

Khi thi hành, phân phát bội đòi hỏi công phu hơn. Theo trực giác, đòi hỏi rằng một phân phát bội buộc phải hoàn thiện và TĐ phân phát bội buộc phải được sắp xếp theo thời gian hoàn thiện phân phát bội trước khi phân phát tới QT ứng dụng. Điều đó tạo nên lý do kết hợp quảng bá nguyên tử với quảng bá thứ tự tổng thành một giao thức. Điều này đưa đến khái niệm phân phát bội thứ tự tổng hai pha. Trong pha đầu tiên của giao thức phân phát bội, QT khởi tạo quảng bá TĐ và thu thập xác nhận với tem thời gian lôgic từ tất cả các thành viên trong nhóm. Suốt thời gian pha 2, sau khi đã thu thập xong mọi xác nhận với tem thời gian lôgic, QT khởi tạo gửi một TĐ cam kết mang tem thời gian xác nhận cao nhất như là thời gian logic đối với việc cam kết. Thành viên trong nhóm sau đó quyết định hoặc TĐ cam kết được đưa vào vùng đệm hoặc phân phát dựa trên thời gian cam kết lôgic toàn cục của TĐ phân phát bội.

Giao thức phân phát bội 2 pha được biểu diễn trong hình 4.8. Trong hình vẽ, hai TĐ, m_1 và m_2 từ hai nguồn khác nhau được quảng bá tới một nhóm. Để rõ ràng, ở đây có hai nguồn (s_1, s_2) và hai thành viên trong nhóm (g_1, g_2). Thời gian đồng hồ lôgic khởi tạo của chúng cho trong vòng tròn. Các đường liền nét và rời nét tương ứng trình bày TĐ và TĐ xác nhận. Mỗi một cung được gán nhãn bởi một cặp hai số. Số đầu tiên (từ 1 đến 8) chỉ bước theo thứ tự bộ phận của xuất hiện và số thứ hai là tem thời gian của TĐ. Ví dụ, QT 1 phân phát bội s_1. Khi mọi xác nhận (bước 2 và 8) đã được s_1 nhận, bộ xử lý tính toán tem thời gian cam kết (9, là lớn nhất của 6 và 9) và trả lại TĐ cam kết cho toàn nhóm. TĐ cam kết mang thời gian hoàn thiện cuối cùng của quảng bá TĐ không được chỉ trong hình. Tương tự, s_2 tính toán tem thời gian cam kết là 8 đối với phân phát bội m_2 của nó. Bảng chỉ dẫn vùng đệm được quản lý bởi CT điều khiển TT của thành viên nhóm g_1. Bộ xử lý đã xác nhận 2 TĐ với tem thời gian là 6 và 8. TĐ cam kết với tem thời gian 8 và 9 có thể tới với thứ tự bất kỳ nhưng CT điều khiển bắt buộc phải chờ cả hai trước khi phân phát được thực hiện. TĐ m_2 được hoàn thiện trước m_1 bởi vì tem cam kết của nó nhỏ hơn. TĐ m_3 (phân phát bội bởi một nguồn khác) không được chú ý tại đây vì TĐ cam kết của nó có tem thời gian cao hơn 10 và như

vậy bắt buộc được phân phát sau m_1 và m_2. Mọi TĐ sau này cũng có tem thời gian lớn hơn và không cần chú ý.



TĐ phân phối bội	Thời gian xác nhận	Thời gian cam kết
m0	2	đã phân phát
m1	6	9
m2	8	8
m3	10	sắp xảy ra

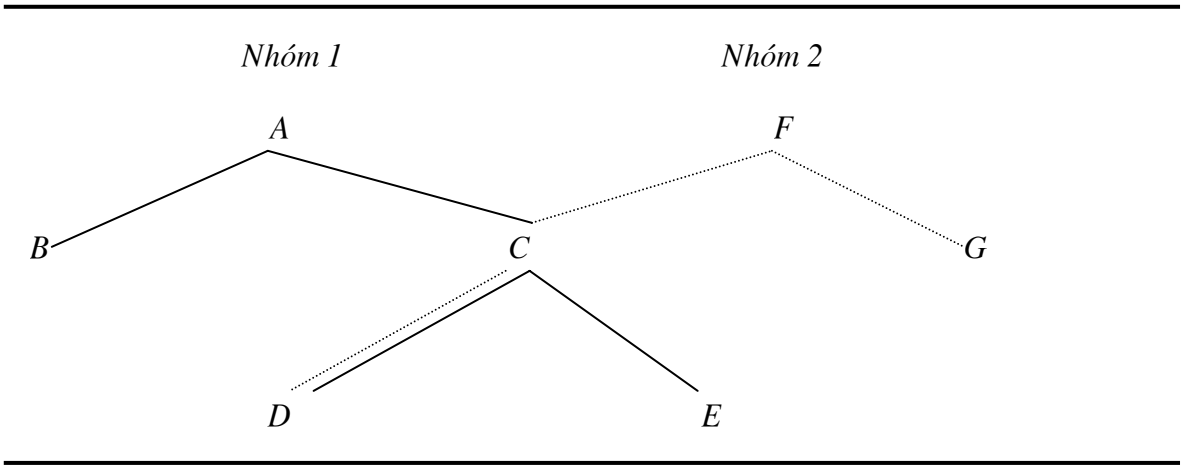
Quản lý vùng đệm trong bộ điều khiển TT

Hình 4.8. Phân phát bội thứ tự tổng hai pha

Bộ đếm TĐ tổng trong giao thức phân phát bội thứ tự tổng hai pha là cao. Nhiều hệ thống (chẳng hạn, ISIS) đơn giản giải pháp thứ tự TĐ tổng bởi giả thiết tồn tại một dịch vụ đánh số dãy toàn cục. Mọi TĐ phân phát bội nhận một số tuần tự toàn cục từ ***bộ sắp xếp dãy***, một bộ xử lý là một thành viên của nhóm. Khi bộ xử lý nhận một TĐ thứ tự tổng, sự phân phát TĐ được làm trễ tới khi số hiệu dãy toàn cục đã được nhận. Bộ sắp xếp dãy đặt vào vùng đệm thứ tự tổng phân phát bội mà nó nhận, gán cho chúng số dãy toàn cục và sau đó phân phát bội số dãy này tới các thành viên khác của nhóm (cần chứng tỏ năng lực gán nhiều số hiệu dãy trong một TĐ đơn là tối ưu). Mỗi khi nhận được số hiệu dãy của phân phát bội toàn cục, bộ xử lý phân phát bội theo thứ tự cho bởi số hiệu dãy toàn cục. Nếu bộ xử lý dãy bị lỗi, một bộ xử lý dãy khác được chọn từ các thành viên trong nhóm.

Trong nhiều ứng dụng phân tán, một QT có thể thuộc vào nhiều nhóm. Hình 4.7.c chỉ ra hai ví dụ tương đương của phân phát bội tới các nhóm giao nhau. Trên đây cho giao thức đánh thứ tự TĐ trong một nhóm đơn. Tuy nhiên, thứ tự có thể khác nhau khi các nhóm rời rạc thậm chí với cùng một TĐ phân phối bội. Với nhóm giao nhau, thì cần phải có sự cộng tác trong nhóm để duy trì thứ tự tường minh của TĐ đối với các thành viên thuộc vùng giao. Một ví dụ về nhóm giao nhau hữu dụng là thi hành các phục vụ được nhân bản khi dùng phân phát bội nguyên tử. Một nhóm chứa chỉ các phục vụ. Với mỗi khách, tồn tại một nhóm khách gồm khách đó và tất cả các phục vụ. Khách có thể thuộc vào một nhóm khác mà chứa các khách khác.

Một giải pháp cho bài toán nhóm giao nhau là đặt cấu trúc được công nhận trên đây đối với nhóm và phân phát bội TĐ sử dụng các cấu trúc này. Ví dụ, các thành viên của nhóm có thể được cấu trúc như là một cây thác triển (cây thác triển là một biểu diễn hợp lý của quan hệ thành viên nhóm trong mạng máy tính không có hỗ trợ quảng bá về phần cứng). Gốc cây đóng vai trò đứng đầu nhóm. Cung của cây trình bày kênh TT FIFO. Một TĐ phân phối bội trước hét gửi tới đỉnh đứng đầu (gốc) và sau đó gửi tới mọi thành viên trong nhóm theo lộ trình TĐ dọc theo các cung của cây. Thành viên trong phần giao phải được cấu hình thành một cây con chung giữa hai nhóm giao nhau. Trong ví dụ hình 4.9. chỉ ra hai nhóm: nhóm 1 gồm các thành viên A, B, C, D và nhóm 2 gồm các thành viên C, D, F và G. Tập giao {C, D} được cấu trúc như một cây con chung giữa hai nhóm.



Hình 4.9. Biểu diễn cây của nhóm giao nhau (liền nét: nhóm 1, rời nét: nhóm 2)

Đạt được sự mềm dẻo hơn nếu như phân phát bội tới nhiều hơn một nhóm (hình 4.7.d). Để đạt được tính nhất quán giữa các nhóm, cần phải xác định một nhóm mới là hợp nhất của hai nhóm. Hình 4.7 b và 4.7.c đã rút gọn vấn đề này.

4.2 Truyền thông hỏi/đáp

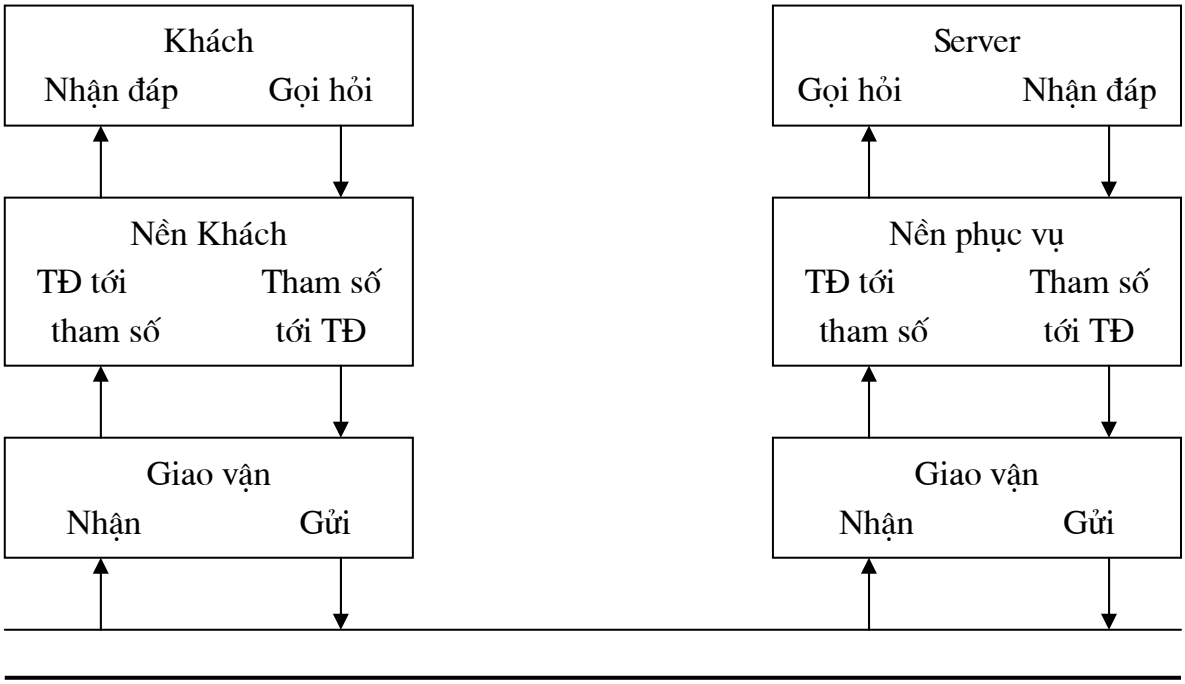
Mức TT ngay trên TT CTĐ cơ sở là TT hỏi/đáp định hướng dịch vụ. Mô hình TT hỏi/đáp được dùng rộng rãi nhất là Lời gọi thủ tục từ xa RPC. RPC là việc trừu tượng ngôn ngữ cơ chế TT hỏi/đáp dựa trên CTĐ. Mục trước đã khẳng định vai trò của RPC trong việc ĐB và TT trong hệ phân tán, còn ở mục này là vấn đề thi hành lời gọi thủ tục từ xa.

4.2.1. Các thao tác RPC

Như thông thường, thao tác gọi thủ tục và chờ kết quả là tương tự cặp TT hỏi/đáp đồng bộ. Điều tương tự giữa lời gọi thủ tục và TT là động lực thúc đẩy nguyên thủy khi dùng lời gọi thủ tục như trừu tượng mức cao cho TT. Một RPC có dạng một lời gọi thủ tục thông thường với các tham số input và output phù hợp của nó. Do không có phân biệt về cú pháp giữa lời gọi thủ tục từ xa và một lời gọi thủ tục cục bộ nên RPC cung cấp sự truy cập trong suốt tới các thao tác từ xa. Tuy nhiên, ngữ nghĩa của chúng là khác nhau do thực hiện thủ tục từ xa bao hàm độ trễ và lượng lỗi có thể trong thao tác mạng. Ứng dụng người dùng cần biết về sự khác biệt này và mối liên quan của chúng. Tuy vậy nhưng RPC là cách đơn giản và trong sáng hoàn thành được tính trong suốt TT bằng cách che dấu lời gọi hệ thống mức thấp, sự biến đổi dữ liệu và TT mạng từ ứng dụng người dùng. Như đã biết (chương II) RPC hỗ trợ một dịch vụ trình diễn giữa tầng giao vận và tầng ứng dụng. RPC có thể được chú ý như một API đối với dịch vụ giao vận. Thao tác RPC cơ sở trong mô hình Client/Server được chỉ ra trong hình 4.10.

Mô tả ngắn gọn về thi hành lời gọi thủ tục từ xa. Giả thiết rằng thông tin cần thiết cho kết nối RPC đã được khởi tạo giữa khách và phục vụ như trong hình 4.10. Lời gọi thủ tục từ xa được khởi tạo từ khách thông qua một lời gọi ***request***, được kết nối với thủ tục nền khách tại nền khách. Thủ tục nền khách chịu trách nhiệm đóng gói lời gọi và tham số của nó thành một TĐ để truyền (điển hình sử dụng API socket) dọc theo mạng nhờ dịch vụ giao vận. TĐ này được dịch vụ giao vận phục vụ tiếp nhận và dịch vụ này chuyển nó tới nền phục vụ. Nền phục vụ là điểm vào chính của phục vụ. Nó tách TĐ thành một lời gọi hỏi với các tham số tương ứng và kích hoạt thủ tục tại phục vụ. Khi hoàn thiện dịch vụ, thủ tục phục vụ đưa lời đáp tới nền phục vụ để đóng gói các tham

số thành một TĐ và gửi nó tới dịch vụ giao vận. Quá trình nhận được thực hiện tại phía khách, và được kết thúc bằng việc nhận được trả lời và loại bỏ việc thực hiện lời gọi.



Hình 4.10. Dòng lời gọi từ xa

Thao tác RPC cơ sở nảy sinh một số vấn đề đáng chú ý sau đây:

• *Truyền tham số và biến đổi dữ liệu*: Kiểu dữ liệu được truyền và dữ liệu được trình bày trong TĐ theo cách nào ?

• *Liên kết*: Làm thế nào khách có thể định vị được phục vụ và bằng cách nào phục vụ ghi nhận được dịch vụ của nó (tạo ra dịch vụ có thể nhìn được từ xa) ?

• *Biên dịch*: Thủ tục nền đến từ đâu và làm cách nào chúng liên kết tới QT khách và QT phục vụ?

• *Loại bỏ và kiểm soát lỗi*: Làm cách nào để kết xuất lỗi và giả thiết nào cần có về lỗi trong hệ thống ?

• *An toàn*: RPC an toàn ?

Ba vấn đề đầu tiên được trình bày như dưới đây.

Truyền tham số và biến đổi dữ liệu

Quy tắc truyền tham số và biến đổi dữ liệu/TĐ RPC được coi là việc sắp xếp lại tham số. Sắp xếp tham số là trách nhiệm nguyên thủy của thủ tục nền. Tồn tại nhiều ngữ nghĩa cho việc truyền tham số đối với lời gọi thủ tục trong ngôn ngữ lập trình bậc cao hiện hành. Đó là các cách thức *gọi theo giá trị*, *gọi theo tên*, *gọi theo chỉ dẫn*, và *gọi qua sao chép/khôi phục*. Phương pháp ***gọi theo giá trị*** trong RPC là trực tiếp. Một giá trị được truyền cho thủ tục được sao vào một biến cục bộ tại điểm vào của thủ tục. Sự thay đổi của biến cục bộ trong lời gọi thủ tục không ảnh hưởng đến lời gọi thủ tục. ***Gọi theo tên*** đòi hỏi việc đánh giá biểu thức ký hiệu trong khi thực hiện động là không thực sự dễ dàng trong môi trường biên dịch. ***Truyền tham số theo chỉ dẫn*** chuyển con trỏ địa chỉ sẽ gây nên sự lúng túng nếu không giảm ngữ nghĩa trong hệ phân tán với hoàn cảnh là không có bộ nhớ trong chia xẻ. Bởi vậy, gọi theo chỉ dẫn không phải là

phương pháp truyền tham số thích hợp đối với RPC. ***Gọi theo sao chép/khôi phục*** kết hợp của gọi theo giá trị và gọi theo chỉ dẫn. Nó là gọi theo giá trị tại điểm vào của lời gọi thủ tục và gọi theo chỉ dẫn có hạn chế tại điểm ra của RPC. Kết quả được sao lại trở lại cho thủ tục gọi; khi hoàn thành thủ tục gọi không tạo ra bất kỳ chỉ dẫn bộ nhớ nào trong quá trình thực hiện thủ tục. Cách thức này có thể được dùng để nắm giữ các con trỏ nhằm đơn giản hóa các cấu trúc dữ liệu kiểu mảng. Cấu trúc con trỏ phức tạp, chẳng hạn như cây và đồ thị, sẽ khó thi hành RPC với mục tiêu nắm giữ mà không cần công sức và phí tổn nào đó. Đa số các thi hành RPC giả thiết tham số được truyền là *gọi theo giá trị* và *gọi theo sao chép/khôi phục*.

Dữ liệu trong ngôn ngữ bậc cao thường được định kiểu theo cấu trúc xác định-tốt. Kiểm tra kiểu tĩnh được trình biên dịch thực hiện khi đối sánh phù hợp hóa kiểu giữa thủ tục nền với khách hoặc phục vụ. Kiểm tra kiểu xuyên qua các máy là khó khăn hơn vì dữ liệu được chuyển thông qua TĐ liên chương trình. Vì vậy, một câu hỏi được nảy sinh là có cần hay không dữ liệu mang kèm theo thông tin kiểu để kiểm tra kiểu động? Hơn nữa, mỗi máy tính lại có cách trình bày dữ liệu riêng của mình. Ví dụ, kiểu integer có thể được trình bày dạng phần bù 2 trong một máy 32 bit song lại có thể là dạng có dấu với lượng 16 bit trong một máy khác. Đối với văn bản, một số máy dùng mã ASCII trong khi một số máy khác dùng EBCDIC. Sự khác nhau này do tính hỗn tạp các thành phần trong hệ thống tạo ra tính cần thiết phải biến đổi dữ liệu trong truyền thông ngang hàng. Tình huống rắc rối hơn khi xem xét việc trình bày chuỗi bit và byte trong kênh truyền thông. Nói riêng, các máy khác nhau có chuẩn khác nhau để các bit hoặc byte trong TĐ được truyền ít nhất hoặc hầu hết chữ số có dấu được truyền trước.

Quy tắc liên quan tới giao vận TĐ trong mạng được gọi là *cú pháp giao vận*. Một số chuẩn biến đổi dữ liệu tường minh bắt buộc được công nhận trong mọi hệ thống khi trình bày dữ liệu (hoặc CSDL) hỗn tạp. Nếu như n cách trình bày dữ liệu thì phải có n*(n-1)/2 cách biến đổi dữ liệu. Giải pháp tốt hơn là tạo ra một ngôn ngữ vạn năng hoặc bộ biểu diễn dữ liệu hợp chuẩn mà mỗi QT TT cần dịch đối với ngôn ngữ hoặc biểu diễn dữ liệu riêng của nó. Rút gọn này cho phép chỉ cần 2*n phép biến đổi đối với hệ thống n cách trình bày. Đáng tiếc là việc sử dụng một ngôn ngữ vạn năng đòi hỏi phải tăng thêm nhiều chi phí về đóng gói và tách gói. Vì vậy, một số nhà sản xuất đề xuất là ngôn ngữ vạn năng được định danh bằng ngôn ngữ bản địa của máy tính do hãng chế tạo. Điểm tối ưu ở chỗ ngăn ngừa được việc dịch nếu các QT TT có thể ngầm định rằng chúng chia xẻ cùng một dạng tự nhiên. Ba vấn đề đáng chú ý trong chuyển đổi dữ liệu tới TĐ và từ TĐ tới dữ liệu như bàn luận trên đây là *định kiểu dữ liệu*, *biểu diễn dữ liệu* và *cú pháp giao vận dữ liệu*.

Một trong những phát triển quan trọng nhất nhằm chuẩn hóa việc định kiểu và biểu diễn dữ liệu là *Bộ chú giải cú pháp trừu tượng 1* (ASN.1). ASN.1 là một ngôn ngữ định nghĩa cấu trúc dữ liệu và được sử dụng rộng rãi để đặc tả khuôn dạng các giao thức chỉnh thể dữ liệu trong TT mạng. Cú pháp giao vận và ASN.1 là điều kiện chính để xây dựng dịch vụ trình diễn mạng. ASN.1 có thể được dùng trực tiếp trong trình diễn dữ liệu để thi hành RPC. Các thi hành RPC hiện tại thường dùng một tập con của ASN.1. Nếu RPC được hỗ trợ trong một miền đơn thì nền khách và nền phục vụ là cộng tác mật thiết. Kiểu dữ liệu được kiểm tra khi sinh và dịch các thủ tục nền. Khi đó không cần cung cấp thông tin kiểu trong TĐ (tức là kiểu đã tường minh trong ASN.1). Trong hệ hỗn tạp, vấn đề liên quan đến cú pháp giao vận được bỏ qua. Các ví dụ kinh điển về ngôn ngữ mô tả và trình diễn dữ liệu đối với RPC là XDR (eXternal Data Representation) của Sun và IDL (Interface Definition Language) của DCE. Cả hai tương tự với ASN.1 song định nghĩa cấu trúc dữ liệu và giao diện thủ tục là đơn giản hơn.

Liên kết

Phục vụ buộc phải tồn tại trước khi khách tạo ra một lời gọi thủ tục tới nó. Dịch vụ này được đặc tả bằng một giao diện phục vụ khi dùng một ngôn ngữ định nghĩa giao diện, chẳng hạn XDR. Một đặc tả giao diện phục vụ điển hình có khuôn dạng được trình bày như hình 4.11. Ví dụ này mô tả hai thủ tục và được định danh duy nhất qua chương trình và số hiệu phiên bản của nó. Khách có thể định vị phục vụ bằng việc quảng bá yêu cầu đối với dịch vụ. Tuy nhiên, giải pháp hiệu quả hơn là đi tới tên riêng phục vụ mà địa chỉ của nó đã được định vị tốt, nếu điều đó là cho phép.

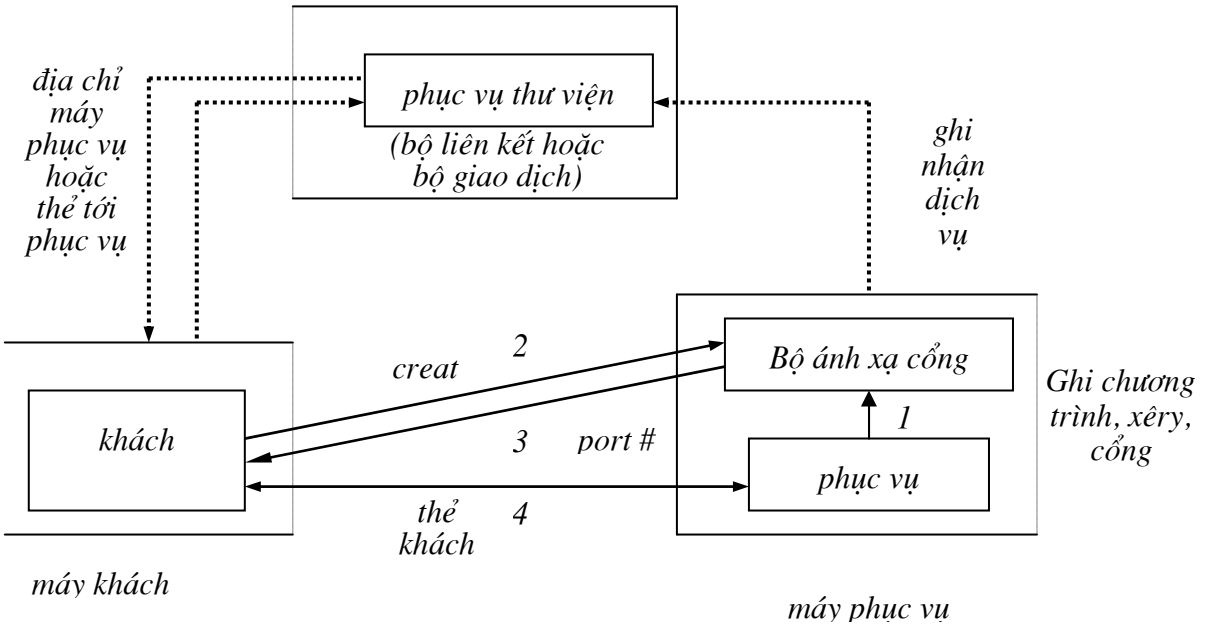
```
program PROGRAMME {
 version VERSIONNAME {
 long PROCEDUREA (parameters) = 1;     /* procedure number = 1 */
 string PROCEDUREB (parameters) = 2;   /* procedure number = 2 */
 } = 1 ;                                /* version number = 1       */
} = 12345 ;                                 /* program number = 12345 */
```

Hình 4.11. Một đặc tả giao thức phục vụ

Hình 4.12 minh họa việc liên kết giữa khách và phục vụ. Liên kết đó được giải thích qua các bước sau đây:

1. Khi phục vụ được khởi động, nó ghi nhận nút TT của mình bằng việc gửi một yêu cầu tới *bộ ánh xạ cổng*. Yêu cầu này bao gồm chương trình của phục vụ, số hiệu phiên bản cùng với số hiệu cổng mà phục vụ dùng để nghe (listen). Bộ ánh xạ cổng quản lý việc ánh xạ giữa số hiệu chương trình và số hiệu cổng. Giả thiết rằng ánh xạ cổng là một QT phục vụ chạy ngầm (daemon) với địa chỉ cổng đã được biết tốt.
2. Trước khi tạo một lời gọi thủ tục từ xa, QT khách bắt buộc tiếp xúc với bộ ánh xạ cổng của hệ thống từ xa để thu được một thẻ (handing) truy nhập tới phục vụ với chương trình riêng và số hiệu phiên bản. Điều này đạt được nhờ việc gọi một chương trình con (routine) ***creat*** của thư viện thời gian chạy RPC và ***creat*** gửi một TĐ chứa tên máy phục vụ, chương trình và số hiệu phiên bản, cùng giao thức giao vận (UDP hoặc TCP) tới bộ ánh xạ cổng từ xa.
3. Bộ ánh xạ cổng qua kiểm tra chương trình và số hiệu phiên bản trong bảng của nó để cung cấp (trả lại) số hiệu cổng của phục vụ tới hệ thống khách.
4. Hệ thống khách xây dựng một thẻ khách cho QT khách để sử dụng sau này trong lời gọi thủ tục từ xa. Quá trình liên kết thiết lập các kết nối socket giữa khách và phục vụ.

Trong trường hợp tổng quát hơn khi chưa biết được máy phục vụ, khách cần định vị máy phục vụ bằng cách tiếp xúc với một phục vụ thư viện (đôi lúc được gọi là ***bộ liên kết***, ***bộ giao dịch***) để định vị địa chỉ của hệ thống phục vụ. Các đường nét rời trong hình 4.12 trình bày thao tác bổ sung này.

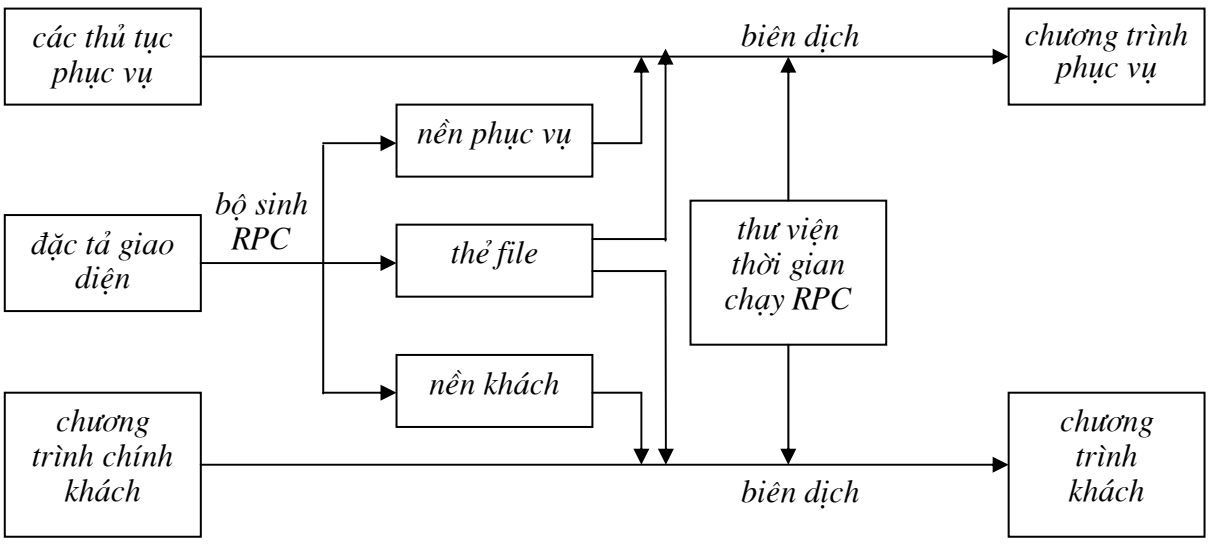


Hình 4.12. Liên kết khách và phục vụ

Biên dịch RPC

Biên dịch các RPC đòi hỏi ba thành phần chính trong bó RPC:

- Một file đặc tả giao diện;
- Một bộ sinh RPC có chức năng nhận input là file đặc tả giao diện và sản xuất ra output là mã nguồn các thủ tục nền khách và phục vụ;
- Một thư viện thời gian chạy để hỗ trợ việc thực hiện RPC bao gồm cả hỗ trợ việc liên



Hình 4.13. Sinh và dịch chương trình RPC

kết, biến đổi dữ liệu và truyền thông.

Hình 4.13 trình bày công việc sinh các thủ tục nền và biên dịch chương trình RPC. Bộ sinh RPC khởi tạo các thủ tục nền và một thẻ file vì tính biên dịch độc lập của các chương trình khách và phục vụ. Do cả thủ tục nền phục vụ và thủ tục nền khách được sinh ra bởi cùng một file giao diện cho nên chúng phù hợp cú pháp với nhau. Mã ghi nhận một phục vụ được chứa trong nền phục vụ như là phần khởi động của QT phục vụ. Lời gọi hệ thống liên kết một khách tới phục vụ này được cho trong QT khách. Lời gọi hệ thống được thực hiện trong lần đầu tiên khi khách có lời gọi RPC tới phục vụ.

4.2.2. Kiểm soát loại bỏ và lỗi RPC

Dù tương tự về khái niệm và cú pháp, RPC vẫn khác với lời gọi thủ tục cục bộ do hạn chế mạng và lỗi. Hai vấn đề cơ bản, loại bỏ và lỗi, cần được định vị đối với thi hành RPC. Loại bỏ là các trạng thái khác thường xảy ra khi thực hiện các thủ tục nền và thủ tục phục vụ. Lỗi là vấn đề gây ra bởi sự đổ vỡ của khách, phục vụ hoặc mạng TT.

Kiểm soát loại bỏ

Loại bỏ trong thủ tục phục vụ, chẳng hạn như overflow/underflow hoặc vi phạm bảo vệ trong khi thực hiện thủ tục bắt buộc phải được kết xuất tới khách. Theo một nghĩa khác, một khách hoặc một thủ tục nền của nó có thể dừng việc thực hiện một thủ tục phục vụ. Những câu hỏi cơ bản là:

- Bằng cách nào phục vụ kết xuất thông tin trạng thái tới khách ?
- Bằng cách nào khách gửi thông tin điều khiển tới phục vụ ?

Các bài toán này được giải quyết dễ dàng theo quy ước trong lời gọi thủ tục cục bộ nhờ sử dụng biến toàn cục chia xẻ và ***tín hiệu***. Ví dụ, trong UNIX biến toàn cục ***errno*** được dùng để kết xuất trạng thái sai sót, và các tín hiệu (hoặc ngắt) có thể được dùng để thực hiện điều khiển đối với các QT khác.

Trên mạng máy tính, không thể sử dụng cả biến chia xẻ lẫn ngắt trực tiếp. Trao đổi điều khiển và trạng thái bắt buộc phải nhờ cậy vào kênh dữ liệu. Tình huống này tương tự vấn đề trong TT dữ liệu khi kênh tín hiệu buộc phải hoặc tìm vị trí của mình trong kênh dữ liệu chuẩn (băng tín hiệu-lõm) hoặc sử dụng một kênh riêng (băng tín hiệu-lồi). Nhiều dịch vụ giao vận cung cấp lựa chọn dữ liệu cờ như điểm lồi của băng thông tin trong ***dịch vụ nguyên thủy gửi***. Cũng có thể dùng kênh riêng (kết nối socket) để khử bỏ đòi hỏi nhận biết tín hiệu từ dữ liệu chuẩn. Tiếp cận như vậy là mềm dẻo hơn đối với RPC do đa số thi hành hệ thống sẵn có giả thiết cổng bội đối với mỗi QT (với mục đích tương tự, chẳng hạn như TT tới nhân). Trong những trường hợp khác, việc gửi và nhận thông tin điều khiển và trạng thái được thi hành như một phần của hỗ trợ thư viện nền và cần phải trong suốt tới QT khách.

Kiểm soát lỗi

Khả năng lỗi xảy ra từ lời gọi thủ tục từ xa là khác với lời gọi thủ tục cục bộ. Việc kiểm soát lỗi để che dấu và trong suốt cho người dùng RPC là một công việc nặng nề. Lỗi xảy ra khi khách không thể định vị được phục vụ tại thời điểm khởi tạo lời gọi RPC. Điều đó xảy ra do phục vụ không tồn tại hoặc phục vụ bị đổ vỡ. Lỗi cũng xảy ra khi khách đang dùng một chương trình hoặc số hiệu phiên bản lỗi thời. Vấn đề này tương tự với loại bỏ hơn là lỗi và có thể được kết xuất như loại bỏ. Mỗi khi phục vụ được định vị và TĐ hỏi đã được gửi tới phục vụ, TĐ có thể bị trễ hoặc mất. Tương tự, TĐ đáp có thể bị trễ hoặc mất. Việc mất TĐ được phát hiện do thời gian quá hạn hoặc không có lời đáp từ phục vụ. TĐ bị trễ hoặc mất có thể được truyền lại.

- Sự truyền lại câu hỏi (từ khách) lại là nguyên nhân của kiểu bài toán khác. Nếu câu hỏi không mất mà đơn thuần chỉ là bị làm chậm, thì phục vụ nhận được hai câu hỏi từ phía khách trong khi mong đợi chỉ một câu hỏi đến đó. Một giải pháp cho vấn đề này

là tạo ra tính bất biến của câu hỏi theo nghĩa câu hỏi được thực hiện với số lần bất kỳ đều nhận cùng một hiệu quả. Ví dụ hệ thống file NFS cung cấp chỉ một dịch vụ bất biến (ví dụ, đọc khối, ghi giá trị vào khối, song không gắn một khối vào một file). Không phải mọi dịch vụ đều có tính bất biến (chẳng hạn như phục vụ khóa). Trong trường hợp này, khách gắn một số hiệu dãy vào mỗi câu hỏi để cho phục vụ có thể phát hiện ra sự đúp hoặc lỗi thời của TĐ hỏi. Nếu phục vụ nhận được sự đúp, nó truyền lại kết quả được kết xuất từ yêu cầu đầu tiên. Chú ý rằng, phục vụ cần giữ lại vết của số hiệu dãy của yêu cầu cuối cùng của mỗi khách và kết quả sinh ra đối với yêu cầu đó. Số hiệu dãy đối với lời gọi RPC khách là không thật sự cần thiết nếu RPC được chạy theo dịch vụ giao vận TCP hướng kết nối tin cậy do dịch vụ đã sắp xếp TĐ và loại trừ sự đúp.

Thi hành điển hình một RPC không dùng đến số hiệu dãy. Phục vụ không chú ý đến khách là ai, có bao nhiêu khách tạo ra câu hỏi như thể là nó đã có cách phát hiện sự đúp. Ví dụ, một giao thức UDP thi hành RPC trên máy Sun dùng một số ngẫu nhiên duy nhất ***xid***, được gọi là số hiệu giao dịch hoặc ***nonce*** (số được dùng chỉ một lần) cho mỗi TĐ hỏi (giao dịch). **xid** được dùng nhằm liên kết hỏi và đáp. Phục vụ RPC duy trì một bảng cache được chỉ số hóa theo chương trình và số hiệu phiên bản, số hiệu thủ tục, địa chỉ UDP khách và **xid** cho mỗi giao dịch hoàn thiện. Kết quả của lần gọi trước đây được trả cho người gọi nếu câu hỏi mới đã có sẵn trên cache. Nếu TĐ đáp là đúp trong bất kỳ lý do nào thì **xid** được khách dùng để loại bỏ sự đúp hoặc thậm chí cả lời đáp sai sót.

• Đỗ vỡ phục vụ là bài toán nguy kịch hơn. Ngay cả khi kết nối TCP tin cậy, thì câu hỏi đúp cũng có thể được phân phát tới phục vụ. Vấn đề này có thể xuất hiện nếu khách đã chờ lời đáp cho câu hỏi của nó đã quá hạn (chẳng hạn, vượt quá hạn TCP). Khách cố gắng thiết lập lại kết nối. Khi kết nối được thiết lập lại (có thể sau khi phục vụ khôi phục lại từ lỗi), khách truyền lại lời hỏi của mình. Chú ý rằng, lỗi của kết nối TCP không có nghĩa là phục vụ bị đỗ vỡ vì rằng trong vài trường hợp kết nối TCP bị đứt do vấn đề mạng, tràn vùng đệm ... Có chăng khi phục vụ bị lỗi, dịch vụ có thể được thực hiện hoặc không. Nếu kết nối TCP bị đứt nhưng phục vụ không bị đổ vỡ, bảng cache được kiểm tra để xác định yêu cầu có đúp hay không. Nếu phục vụ bị đổ vỡ, bảng cache bị mất. Trong trường hợp này, phục vụ có thể chọn đề xuất một loại bỏ tại QT khách và QT khách hoặc chờ cho phục vụ khôi phục lại hoặc từ bỏ ngay lập tức. Từ đó dẫn đến ba giả thiết khả năng cho ngữ nghĩa lời gọi RPC khi hiện diện lỗi:

• phục vụ đề xuất một loại bỏ và khách thử lại thao tác khi phục vụ hồi phục. Do thao tác sẽ được thi hành ít nhất một lần thì ít nhất phải có một lần ngữ nghĩa. Ngữ nghĩa này được thừa nhận cho thao tác bất biến.

• phục vụ đề xuất một loại bỏ và khách từ bỏ ngay lập tức. Do thao tác yêu cầu có thể đã được thực hiện bởi phục vụ trước khi bị đổ vỡ, tồn tại ít nhất một ngữ nghĩa.

• phục vụ không kết xuất lỗi nào cả, và khách gửi lại yêu cầu của nó cho đến khi nó được đáp hoặc từ bỏ. Điều này cũng có thể ngữ nghĩa do thao tác có thể đã được thực hiện số lượng lần bất kỳ (bao gồm cả không lần nào).

Đương nhiên, ngữ nghĩa RPC mong muốn nhất là một lần chính xác. Khó khăn khi hoàn thành một lần chính xác mà không cần đòi hỏi sự cố gắng. Do vấn đề ở chỗ bị mất bảng cache, giải pháp là *ngữ nghĩa ít nhất một lần* và *chặt đoạn bảng cache lên bộ nhớ ngoài*. Khi phục vụ được hồi phục, nó tải lại bảng cache từ các đoạn của nó. Tuy nhiên, thao tác thích hợp mỗi dịch vụ bắt buộc được thực hiện như một giao dịch tại phục vụ. Điều này đòi hỏi quá nhiều chi phí đối với nhiều ứng dụng.

• Cuối cùng, nếu QT khách đỗ vỡ (hoặc kết thúc vội vã) trước khi phục vụ hoàn thiện yêu cầu của khách, phục vụ có một tính toán ***orphan*** (đơn độc) và đáp của nó là không được phân phát. Không có cách dễ dàng để phục vụ kiểm tra sự biến mất của khách ngoại trừ việc dùng quá hạn hoặc chờ khách lỗi để reboot và loan báo sự hiện diện mới của nó. Tính toán đơn độc tiêu thụ các tài nguyên phục vụ (không gian bộ nhớ, khoá) và cũng có thể làm lộn xộn khách với câu đáp không có giá trị trong kết nối trước. Tính toán đơn độc có thể bị loại trừ theo các cách sau đây:

• Bằng khách: Nhờ vào việc reboot khách lỗi, khách đó làm sạch một cách rõ ràng các tính toán đơn độc trước đây. Đòi hỏi khách nhận thức được hoạt động của lời gọi thủ tục chưa kết thúc trước đây của nó và năng lực định vị các lời gọi đó.

• Bằng phục vụ: phục vụ thỉnh thoảng thử định vị chủ nhân của các thao tác từ xa của nó và bỏ đi những thao tác không tìm thấy chủ nhân. Giải pháp này đòi hỏi sự hoán vị vai trò của khách và phục vụ, phức tạp hóa một thiết kế đơn giản khác.

• Bằng sự kết thúc: Mỗi thao tác được tương ứng với một thời gian sống cực đại. Một thao tác bị loại bỏ khi nó đạt tới thời gian kết thúc của nó (ngoài trừ khách yêu cầu thêm thời gian một cách rõ ràng).

4.2.3. Bảo mật RPC

Bảo mật là vô cùng quan trọng đối với RPC bởi hai lý do:

• RPC là hình thức thực hiện từ xa cho phép chương trình hoặc lệnh được thực hiện tại một hệ thống khác. Nó là phương tiện đầy sức mạnh thi hành hệ thống và các ứng dụng phân tán. Tuy nhiên, RPC lại là một nguồn gốc làm cho hệ thống dễ bị xâm phạm khi người dùng không thân thiện sử dụng RPC để tấn công từ xa.

• RPC đã trở thành nền tảng của tính toán Client/Server. Tất cả các đặc điểm an toàn của hệ thống máy tính sẽ được xây dựng dựa trên an toàn RPC.

RPC dựa trên trao đổi TĐ hỏi/đáp giữa khách và phục vụ. Các vấn đề an toàn nguyên thủy là tính xác thực của QT khách và QT phục vụ, tính xác thực và bí mật của TĐ, và tính xác thực điều khiển truy nhập từ khách tới phục vụ. Vấn đề an toàn máy tính phân tán tổng thể được trình bày ở chương sau. Tại đây trình bày các khái niệm thích hợp về tính xác thực lưu tâm tới RPC. Một giao thức xác thực cho RPC cần được thiết lập như sau:

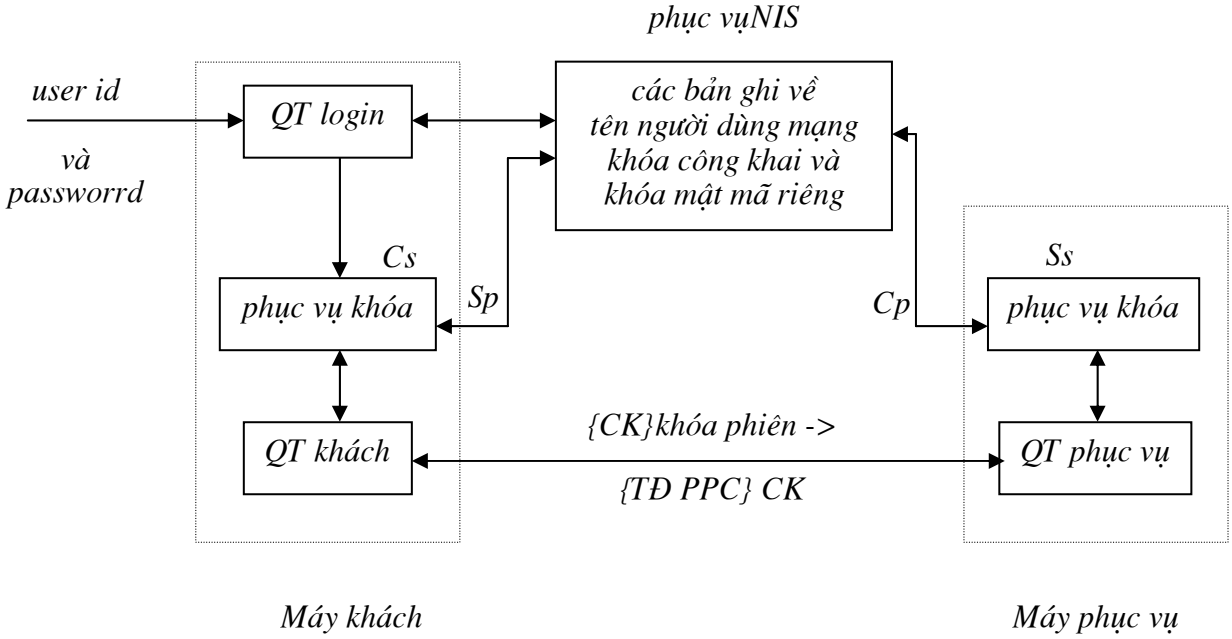
• Xác thực lẫn nhau: Định danh của khách và phục vụ cần được kiểm tra. TĐ hỏi thực sự được sinh ra từ khách và được mong đợi tại phục vụ. TĐ đáp thực sự sinh ra bởi phục vụ và được mong đợi tại khách. Tính xác thực bắt buộc phải được đảm bảo đối với TĐ và đối với các QT TT.

• Tính toàn vẹn, tính tin cậy và tính nguyên bản của TĐ: TĐ hỏi/đáp không bị xáo trộn (tính toàn vẹn), nội dung của nó không bị lộ (tính tin cậy) và một TĐ không xuất hiện quá một lần (tính nguyên bản).

Thiết kế giao thức xác thực là nội dung rất phức tạp. Tính phức tạp của giao thức phụ thuộc vào mục tiêu an toàn cao đến đâu, những tấn công thích hợp nào được đoán nhận, và một số hạn chế cố hữu của hệ thống. Hệ thống an toàn cao đòi hỏi nhiều biến đổi cơ sở trong HĐH. Giải pháp thích hợp hơn mong muốn có được những đặc điểm an toàn dễ dàng bổ sung vào HĐH đang tồn tại. Nội dung dưới đây về RPC an toàn của Sun là ví dụ về tính an toàn có thể kết hợp chặt chẽ dễ dàng vào một HĐH đang tồn tại.

RPC an toàn Sun được xây dựng trong hệ thống RPC cơ sở của Sun. Giả thiết tồn tại dịch vụ thông tin mạng (NIS) tin cậy được đặt tại một phục vụ xác thực được chia xẻ trong hầu hết các giao thức xác thực. Tuy nhiên, NIS trong RPC xác thực Sun không

thực hiện chức năng xác thực. Nó đơn giản được đặt ra để giữ một CSDL chứa các bản ghi về tên mạng của người dùng và khóa công khai và khóa bí mật. Các khóa này không phải là mã hóa/giải mã như chúng thông thường được dùng trong mật mã. Thay vào đó, chúng được dùng để sinh ra một khóa phiên mật mã đúng cho TT. Khóa công khai là thông tin công khai, còn khóa bí mật được tạo bởi mã hóa chúng khi dùng thuật toán mã hóa DES (Data Encryption System) với mật khẩu người dùng như một khóa mật mã. Khi người dùng login máy trạm, RPC an toàn chạy, chương trình login tiếp xúc với NIS để nhận bản ghi khóa của người dùng. Sau đó chương trình login nhắc người dùng nhập mật khẩu, sử dụng mật khẩu để giải mã khóa bí mật đã được mã hóa, và loại bỏ mật khẩu ngay lập tức sau đấy. Pha login người dùng được chỉ ra trong hình 4.14. Chú ý rằng mật khẩu người dùng không truyền trên mạng.



Hình 4.14. RPC an toàn của Sun

Sau khi login thành công, QT khách, làm việc nhân danh người dùng, thiết lập một phiên TT với QT phục vụ như sau. Chương trình login đặt khóa bí mật của khách vào vùng nhớ phục vụ khóa. Thủ tục giống như ở phía khách cũng diễn ra ở phía phục vụ. QT phục vụ khóa tại khách và phục vụ chịu trách nhiệm sinh ra khóa phiên chung giữa khách và phục vụ. Điều đó được thực hiện bằng việc trao đổi khóa đã mũ hóa. Đầu tiên, một cặp khóa riêng và khóa công khai được gán (hoặc ghi nhận) đối với mọi khách {C1, C2, ..., Cp}và mọi phục vụ {S1, S2, ..., Sp} trong hệ thống. Các khóa này được sinh theo con đường sau đây:

- Cs và Ss là các số ngẫu nhiên 128 bit,
- $Cp = \alpha^{Cs} \bmod M$ và $Sp = \alpha^{Ss} \bmod M$, trong đó α và M là hai hằng số đã biết.

Mặc dù khóa công khai thu được từ khoá bí mật song thao tác tìm ngược theo logarith rời rạc được đánh giá là tính toán rất tốn kém. Người dùng không thể dễ dàng suy luận được khóa bí mật từ khóa công khai tương ứng của nó, thậm chí trong trường hợp khi đã biết thuật toán. Tại phía khách, khóa phiên SK_{CS} đươc tính qua khóa bí mật khách Cs và khóa công khai phục vụ Sp như sau:

$$SK_{CS} = Sp^{Cs} = (\alpha^{Sp})^{Cs} = \alpha^{Sp \circ Cs}$$

Hoàn toàn đối xứng và độc lập, phục vụ khóa tính khóa phiên SK_{SC} nhờ khóa bí mật phục vụ Ss và khóa công khai khách Cp như sau:

$$SK_{SC} = Cp^{Sp} = (\alpha^{Cs})^{Sp} = \alpha^{Cs \circ Sp}$$

Có thể thấy rằng, $SK_{CS} = SK_{SC} = SK$. Khoá phiên được tính toán theo MOD M (để rõ ràng bỏ qua toán tử *mod*). Mỗi khi khóa phiên được sinh, khóa bí mật được xoá khỏi vùng nhớ của dịch vụ khóa. Khóa phiên theo thỏa thuận thiết lập xác thực lẫn nhau của khách và phục vụ do nó chỉ nhận được từ khóa bí mật trình diễn định danh thực sự của bộ đôi TT.

Mỗi TĐ RPC được xác thực bởi khóa kết hợp CK là một số ngẫu nhiên 56-bit được khách sinh ra và được truyền tới phục vụ nhờ khóa phiên. Khóa kết hợp được giữ trong phục vụ khóa và được dùng trong toàn bộ phiên giữa khách và phục vụ. Khóa phiên được dùng sơ bộ trong mạng còn khoá kết hợp là được chuyển giao. Khả năng bị làm hại là tối thiểu. Lý do sử dụng khóa kết hợp đối với TĐ RPC kế tiếp là nó không bắt nguồn từ khóa bí mật và bởi vậy, được lưu lại để dùng cho giai đoạn dài. Nó khác nhau trong mỗi phiên.

TĐ RPC có thể chứa thêm nhiều thông tin, bao gồm *tem-thời gian*, (số hiệu) *lượt* và một *tóm tắt T*Đ. Tem-thời gian được dùng để kiểm tra kết thúc TĐ. Lượt được dùng để đảm bảo không xảy ra việc lặp TĐ. Tóm tắt TĐ là một giá trị băm của dữ liệu TĐ được dùng để phát hiện bất kỳ can thiệp nào vào TĐ. Mã hóa thông tin này bằng khóa kết hợp cung cấp cả tính xác thực lẫn tính an toàn của TĐ.

Thi hành RPC của Sun là rất đơn giản. Nó dùng NIS tồn tại thay cho một phục vụ xác thực riêng biệt. Thông tin bí mật được chuyển tới mạng hoặc bảo quản tại máy khách và máy phục vụ được giữ ở mức độ tối thiểu.

4.3. Truyền thông giao dịch

TT hỏi/đáp định hướng dịch vụ và đa phân phát được trình bày trên đây có thể kết hợp lại thành một mức TT mới cao hơn được gọi là TT giao dịch. Phổ biến hơn, giao dịch được biết như một đơn vị chuẩn của liên-hành động giữa QT khách và phục vụ trong hệ CSDL. Giao dịch CSDL được biểu diễn như một dãy thao tác hỏi/đáp đồng bộ đảm bảo tính nguyên tử (không chia cắt được), tính nhất quán, tính cô lập (riêng biệt) và tính bền vững (ACID) như trình bày dưới đây. Giao thức trong TT tương tự như giao thức trong CSDL ngoại trừ chúng được định nghĩa như một tập TT hỏi/đáp ***dị bộ*** có các tính chất ACID nhưng không ép buộc tính tuần tự thao tác như giao dịch CSDL. Một giao dịch TT có thể bao gồm đa phân phát cùng một TĐ tới các phục vụ nhân bản và yêu cầu khác nhau tới phục vụ được phân chia. Dịch vụ giao dịch và kiểm tra đồng thời đối với giao dịch CSDL sẽ được nói sau. Trình bày dưới đây giới hạn trong khuôn khổ TT giao dịch.

4.3.1. Các tính chất ACID

Các tính chất ACID liên quan trước hết tới mục tiêu trong suốt đồng thời của hệ phân tán. Đoạn 2.2 đã giới thiệu trong suốt đồng thời là tính chất cho phép chia xẻ các đối tượng mà không gặp trở ngại. Về cảm giác, thực hiện giao dịch có vị trí như khoảng tới hạn. Tuy nhiên, các thao tác từ các giao dịch khác nhau là được xen kẽ (theo một cách đảm bảo) nhằm tăng tính đồng thời. Nói thêm, giao dịch có các tính chất bổ sung:

- Tính nguyên tử (A: Atom): Hoặc tất cả thao tác được thực hiện hoặc không một thao tác nào trong giao dịch được thực hiện bất chấp lỗi.

- Tính nhất quán (C: Consistency): Thực hiện trộn xen kẽ các giao dịch tương đương với thực hiện tuần tự các giao dịch đó theo thứ tự nào đó.
- Tính cô lập (I: Isolation): Kết quả thực hiện bộ phận giao dịch chưa hoàn thiện được che dấu đối với các giao dịch khác trước khi giao dịch được cam kết thành công.
- Tính bền vững (D: Durability): Hệ thống đảm bảo rằng kết quả thực hiện giao dịch đã cam kết được lưu giữ lâu dài, ngay cả khi xuất hiện lỗi sau cam kết.

Do cả bốn tính chất này quan hệ với tính nhất quán nên trong nhiều trường hợp, thường gọi tính chất thứ hai là tính thi hành dãy để phân biệt với các tính chất khác. Tính nguyên tử cho tính nhất quán của đối tượng khi nhân bản hay phân hoạch. Vi phạm sự cô lập thì thấy được cái không bao giờ xẩy ra, còn vi phạm tính bền vững thì lại không thấy được cái thực tế xảy ra. Cả hai điều này là không nhất quán về trạng thái hệ thống.

Bảo đảm các tính chất ACID đòi hỏi các QT thành phần cộng tác thực hiện giao dịch. Gọi QT khởi tạo giao dịch là bộ phối hợp và các QT còn lại là thành viên. Bộ phối hợp bắt đầu giao dịch bằng một đa phân phát yêu cầu tới các thành viên. Giao dịch kết thúc bằng cam kết hoặc hủy bỏ giao dịch tùy thuộc vào các tính chất ACID có được đảm bảo hay không. Mọi thành viên phải tán thành quyết định cuối cùng. Một giải pháp cho đòi hỏi tính nguyên tử của giao dịch là cần thường xuyên ngăn cản thao tác của mỗi thành viên cho đến khi đã chắc chắn hoặc được thông báo là mọi thành viên khác đã sẵn sàng làm thao tác đó. Kỹ thuật này tương tự như hai giai đoạn (nhận và phân phát TĐ) trong giao thức hai pha đối với đa phân phát thứ tự toàn bộ trong đoạn 4.1.5. Giao thức cam kết hai pha đối với giao dịch nguyên tử cho cách thức để đạt được tính nguyên tử, tính cô lập và tính bền vững.

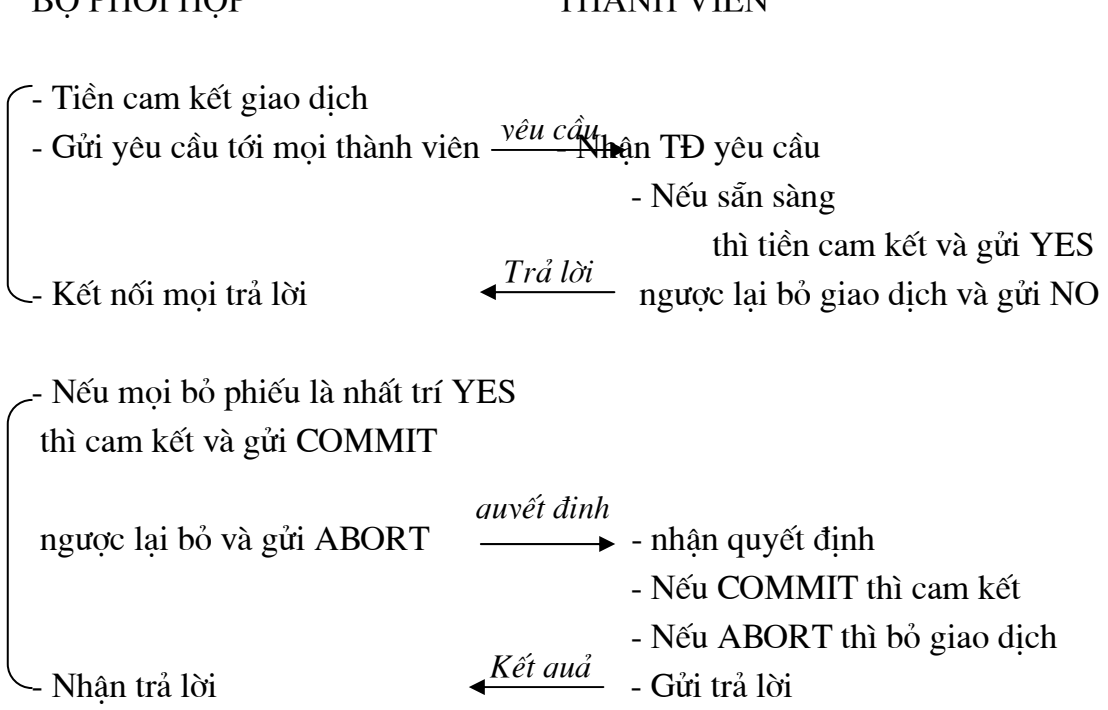
4.3.2. Giao thức cam kết hai pha

Giao thức cam kết 2 pha (2PC) tương tự như sơ đồ bỏ phiếu nhất trí trong cuộc sống đời thường. Bỏ phiếu được bộ phối hợp của giao dịch khởi động. Tất cả các thành viên phải đi tới nhất trí về việc cam kết hoặc hủy bỏ giao dịch và buộc phải đợi thông báo về quyết định. Trước khi thành viên bỏ phiếu cam kết giao dịch, nó cần phải chuẩn bị thực hiện cam kết. Giao dịch được cam kết chỉ khi mọi thành viên đồng ý và sẵn sàng cam kết.

Mỗi thành viên (bao gồm cả bộ phối hợp) duy trì một không gian làm việc riêng để giữ ***vết*** cập nhật đối tượng dữ liệu. Mỗi cập nhật gồm giá trị cũ và giá trị mới của đối tượng dữ liệu. Cập nhật là không thực sự (lâu dài) cho đến khi giao dịch được cam kết cuối cùng nhằm đảm bảo ngữ nghĩa cô lập của giao dịch. Cần đưa các cập nhật lên bộ nhớ lâu dài để đối phó với lỗi. Cập nhật được ghi nhận trong bộ nhớ lâu dài vào *sổ lộ trình hoạt động* của giao dịch. Mỗi thành viên có một sổ lộ trình. Sổ lộ trình được duyệt lại trong khi sửa lỗi nhằm làm dễ dàng hơn công việc hoặc tu sửa lại giao dịch đã cam kết hoặc tháo bỏ giao dịch không cam kết. Sổ lộ trình hoạt động nhất quán là cần thiết cho tính nhất quán hoặc tính bền vững của giao dịch đã cam kết.

Hình 4.15. minh họa dòng thực hiện giao dịch nguyên tử cam kết 2-pha. Tại đây có hai điểm đồng bộ là *tiền cam kết* và *cam kết* đối với mỗi thành viên. Bộ phối hợp bắt đầu giao dịch bằng việc ghi nhận bản ghi tiền cam kết vào sổ lộ trình hoạt động của nó. Bộ phối hợp phải chuẩn bị cam kết giao dịch (tức là, cập nhật được hướng tới sổ lộ trình ổn định, các tài nguyên dành để thực hiện lời cam kết...) trước khi ghi bản ghi tiền cam kết. Ghi bản ghi tiền cam kết vào sổ lộ trình hoạt động cho phép bộ phối hợp biết trạng thái của giao dịch nếu một lỗi xuất hiện; giao dịch kết thúc thực hiện mà nó không cam kết. Sau đó, bộ phối hợp đa phân phát yêu cầu bỏ phiếu tới mọi thành viên. Khi nhận được yêu cầu bỏ phiếu, mỗi thành viên kiểm tra xem có cam kết được giao dịch hay

không (các cập nhật đã được hướng tới sổ lộ trình hoạt động, phát hành đã được tin tưởng, tài nguyên đã sẵn sàng ...). Nếu kiểm tra thấy hợp lý thì thành viên ghi một tiền cam kết vào sổ lộ trình và gửi TĐ YES tới bộ phối hợp. Ngược lại thành viên bỏ giao dịch và gửi TĐ NO tới bộ phối hợp.



Hình 4.15. Giao thức giao dịch nguyên tử cam kết hai pha

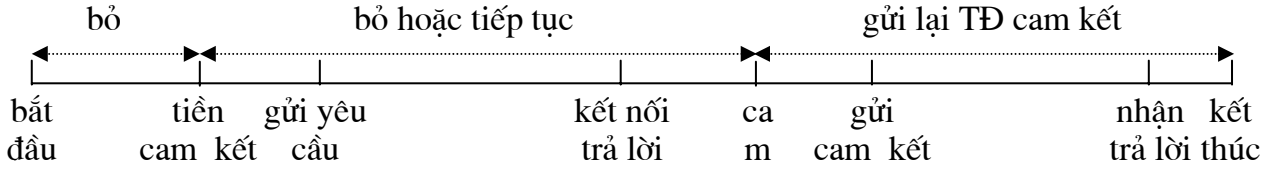
Nếu trong khoảng thời gian quy định, bộ phối hợp kết nối được tất cả các trả lời YES thì nó cam kết giao dịch bằng việc ghi bản ghi cam kết tới sổ lộ trình của nó và đa phân phát TĐ COMMIT tới mọi thành viên. Ngược lại, bộ phối hợp hủy bỏ giao dịch và đa phân phát TĐ ABORT. Khi nhận được TĐ COMMIT, mỗi thành viên cam kết giao dịch bằng việc ghi bản ghi cam kết vào sổ lộ trình hoạt động và tiếp nhận tài nguyên dành cho giao dịch. Cuối cùng gửi một trả lời cho bộ phối hợp. Nếu TĐ nhận được là ABORT thì thành viên ghi bản ghi hủy bỏ vào sổ lộ trình, hủy bỏ giao dịch và giải phóng tài nguyên dành cho giao dịch.

Dùng sổ lộ trình hoạt động, giao thức cam kết 2-pha mạnh mẽ đối với các lỗi QT. Hình 4.16 chỉ dẫn mạch thời gian của giao thức đối với bộ phối hợp và một thành viên. Do việc ghi tiền cam kết và cam kết tới sổ lộ trình hoạt động làm sạch mọi cập nhật trước điểm đồng bộ này, thao tác thích hợp trong khôi phục lỗi tin cậy được là làm lại thao tác theo sổ lộ trình ít nhất từ một điểm đồng bộ. Như vậy, thao tác khôi phục được phân thành ba kiểu: *lỗi trước tiền cam kết*, *lỗi sau tiền cam kết song trước cam kết*, *lỗi sau cam kết*. QT (hoặc bộ phối hợp hoặc thành viên) một cách đơn giản bỏ giao dịch nếu từ sổ lộ trình phát hiện lỗi xuất hiện trước tiền cam kết. Điều đó tương đương với phiếu NO đối với giao dịch. Bộ phối hợp cũng bỏ giao dịch nếu nó sụp đổ giữa tiền cam kết và cam kết, nhưng hiệu quả hơn thì nên cố gắng tiếp tục cam kết giao dịch bằng cách đa phân phát lại TĐ yêu cầu (nhân bản có thể được các thành viên phát hiện ra). Tương tự, thận trọng gửi lại TĐ cam kết nếu bộ phối hợp sụp đổ sau khi ghi bản ghi cam kết vào sổ lộ trình của nó. Thao tác khôi phục sẽ phức tạp hơn chút ít nếu một

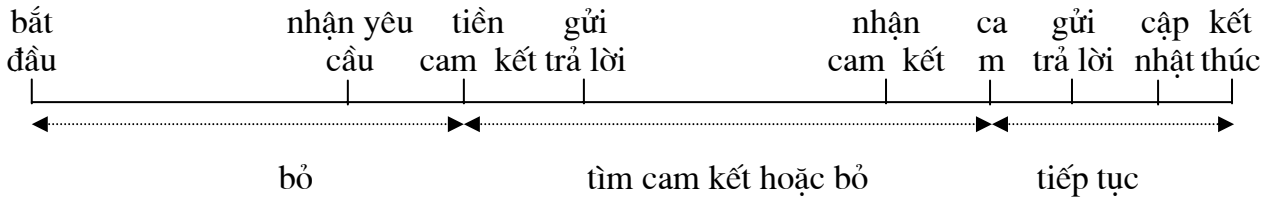
thành viên sụp đổ giữa tiền cam kết và cam kết. Việc phục hồi thành viên bắt buộc phải xác định được giao dịch là cam kết hay bỏ qua hợp đồng thao tác với bộ phối hợp hoặc các thành viên khác. Cuối cùng nếu thành viên khôi phục do lỗi sau khi bản ghi cam kết đã được ghi vào sổ lộ trình thì thành viên đó đơn giản chỉ tạo ra các cập nhật giao dịch lâu dài. Với bộ nhớ cố định, cơ chế khôi phục tin cậy cho tính bền vững của cam kết.

Chương tiếp theo trình bày hình thức hơn giao thức cam kết hai pha và nâng cấp của nó: giao thức cam kết 3-pha. Chương sau đó được dành cho lỗi và phục hồi.

Thao tác khôi phục lỗi của Bộ phối hợp



Thao tác khôi phục lỗi của thành viên



Hình 4.16. Lỗi và thao tác phục hồi đối với giao thức 2-PC

4.4. Dịch vụ tên và thư mục

Khi đề cập tới truyền thông, vấn đề tên và địa chỉ là rất cần thiết. Khi tạo yêu cầu một dịch vụ hoặc truy cập tới một đối tượng theo nghĩa TTLQT đòi hỏi bắt buộc đầu tiên là phải định vị được dịch vụ hoặc đối tượng. Dịch vụ là đối tượng được trừu tượng hóa. Chúng thường được QT trình bày bằng một điểm truy cập dịch vụ. Đối tượng có thể là người dùng, máy tính, đường TT hoặc các tài nguyên khác, chẳng hạn file. Dịch vụ và đối tượng thường được định danh bằng tên kết cấu. Như là một chọn lựa, nếu tên là chưa biết, điểm vào dịch vụ hoặc đối tượng được mô tả thông qua sử dụng thuộc tính phù hợp với chúng. Do dịch vụ hoặc đối tượng có nghĩa rất rõ ràng, vấn đề tên của chúng là tương tự nhau. Khi nghiên cứu về dịch vụ tên và thư mục, dùng thuật ngữ ***mục đối tượng*** để chỉ cả dịch vụ lẫn đối tượng.

4.4.1. Giải pháp tên và địa chỉ

Dịch vụ tên và thư mục, theo nghĩa hẹp là các toán tử tra cứu (look-up). Khi cho một tên hoặc một vài thuộc tính của một mục đối tượng thì thu được thêm nhiều thông tin thuộc tính. Thuật ngữ dịch vụ tên và dịch vụ thư mục thường được sử dụng hoán đổi nhau. Dịch vụ tên là cách tổng quát mô tả một đối tượng được địa chỉ hoá và do đó có thể xác định được đối tượng thông qua địa chỉ. Khái niệm dịch vụ thư mục trong nhiều trường hợp, được dùng để chỉ dịch vụ tên đặc biệt như dịch vụ thư mục của hệ thống file. Trong các trường hợp khác, nó không bị hạn chế bởi thông tin địa chỉ mà còn được dùng để trình bày một dịch vụ đặt tên chung nhất đối với tất cả các kiểu tra cứu thuộc tính trên những loại đối tượng khác nhau. X500 được CCITT định nghĩa là ví dụ cho dịch vụ kiểu này. Những dịch vụ tên mức cao được xây dựng dựa trên dịch vụ thư mục chuẩn.

Mục đối tượng bất kỳ trong hệ thống phải được đặt tên (hoặc định danh) và được định vị trước khi sử dụng. Hoạt động định vị đối tượng được gọi là QT *giải pháp* (*resolution*), cần đến một loạt các thao tác tra cứu (*hoặc ánh xạ)*. Mỗi mục đối tượng có một địa chỉ logic trong HĐH và một địa chỉ vật lý trong mạng. QT *giải pháp* gồm hai phần: *Giải pháp tên* ánh xạ tên tới địa chỉ logic và *Giải pháp địa chỉ* ánh xạ địa chỉ logic tới đường truyền vật lý trong mạng.

Giải pháp tên

Tên là dấu hiệu định hướng ứng dụng của đối tượng. Địa chỉ đại diện cho đối tượng, mang thông tin cấu trúc nào đó thích hợp để HĐH quản lý và định vị đối tượng. QT giải pháp tên ánh xạ tên tới địa chỉ. Đối tượng được xác định bằng tên, và đối tượng được định vị thông qua địa chỉ. Tên thường là duy nhất nhưng cùng một tên có thể có nhiều địa chỉ. Ánh xạ tên một phục vụ tới các địa chỉ cổng của nó được coi là ví dụ về giải pháp tên.

Giải pháp địa chỉ

Giải pháp địa chỉ ánh xạ địa chỉ tới các đường truyền mà theo đó định vị vật lý một đối tượng. Sự khác nhau giữa một địa chỉ và một đường truyền là đường truyền là mức thấp nhất của TT và không còn mức nào thấp hơn, trong khi đó địa chỉ lại bao gồm những thông tin định danh đối tượng trung gian giữa tên và đường truyền. Việc ánh xạ cổng phục vụ tới các cổng Ethernet của nó được coi là ví dụ về giải pháp địa chỉ.

Giải pháp tên là một chức năng cơ bản của HĐH phân tán, trong khi giải pháp địa chỉ là bài toán của mạng. Phần này tập trung vào giải pháp tên, ánh xạ tên tới địa chỉ chuẩn trong HĐH. Dịch vụ tên được thi hành bằng một hoặc nhiều phục vụ tên. Nhấn mạnh đặc biệt là giải pháp tên được gọi thường xuyên và kéo theo một số phục vụ tên.

4.4.2. Thuộc tính đối tượng và cấu trúc tên

Mục đối tượng được đặc trưng bằng các thuộc tính của nó. Ví dụ, người dùng có thể có thuộc tính là địa vị, file có thể có thuộc tính là số hiệu phiên bản, ngày khởi tạo, và các thuộc tính khác. Trong giải pháp tên, quan tâm riêng tới hai thuộc tính đối tượng đặc biệt, đó là *tên* và *địa chỉ*. Tên là thuộc tính với giá trị duy nhất để định danh. Địa chỉ cũng là thuộc tính, ngoại trừ một điều rằng nó là kết quả ra của QT giải pháp tên. Tập hợp tất cả các tên, được tổ chức qua dịch vụ tên cùng các thuộc tính và địa chỉ tương ứng của chúng, được gọi là không gian tên. Một không gian tên rộng lớn chứa các lớp đối tượng khác nhau, có thể được cấu trúc tới các kiểu đơn nhất của nó. Ví dụ, đối tượng có thể được phân lớp thành người dùng, máy tính hoặc file... Những kiểu đối tượng này trở thành thuộc tính để tổ chức không gian tên.

Tên của một mục đối tượng có thể là một thuộc tính đơn giản nhưng cũng có thể chứa nhiều thuộc tính có cấu trúc nội tại hoặc không. Cấu trúc-tên chỉ dựa theo một thuộc tính được gọi là cấu trúc tên phẳng *(flat)*. Một cấu trúc tên-phẳng là một thuộc tính tượng trưng. Yêu cầu duy nhất đối với cấu trúc phẳng là cách đặt tên phải duy nhất trong toàn hệ thống. Cách này là đơn giản nhất và rõ ràng cho tính trong suốt định vị và độc lập. Bất lợi lớn nhất là cách đặt tên này rất khó thực hiện đồng thời trên diện rộng vì rất khó quản lý và cung cấp tên do tên không có cấu trúc.

Nếu tên được phân ra thành nhiều thuộc tính thì có thể lợi dụng thứ tự các thuộc tính. Chẳng hạn, người sử dụng với thuộc tính tên là <A>, thuộc tính tổ chức là <FOT> và thuộc tính quốc gia là <VN> thì có thể tạo ra thành một thuộc tính tổng hợp là <A.FOT.VN> như thuộc tính tên. Theo sơ đồ phổ dụng hơn (dùng trong mạng Internet là Domain Name Service - dịch vụ tên miền), tên là thuộc tính cấu trúc phân cấp. Tổng quát hơn, khi không cho thứ tự các thuộc tính thì có thể định vị đối tượng nhờ tập hợp

tất cả các thuộc tính. Ví dụ U=A, C=FOT, O=VN khi đó U, C, và O đại diện cho các thuộc tính là tên, cơ quan và quốc gia. Những tên này được xác định dựa vào việc ghép nối kế tiếp các thuộc tính (giải pháp này dựa vào tên cấu trúc phân cấp) hoặc dựa vào chỉ tập hợp các thuộc tính (giải pháp dựa vào thuộc tính với cấu trúc tùy ý). Hai giải pháp tên này tương tự như trang trắng (white page) và trang vàng (yellow page) tương ứng trong sổ danh mục điện thoại.

Việc phân chia thuộc tính cho giải pháp tên của đối tượng có thể căn cứ vào tính chất vật lý, theo tổ chức hoặc theo chức năng.

Một cấu trúc tên được phân chia theo vật lý cho phép định rõ thông tin vị trí về đối tượng, ví dụ <user.host.network>. Một cấu trúc tên được phân chia theo cách tổ chức dựa vào cấu trúc hành chính nhà nước. Ví dụ như <user.department.organization>. Cả hai cách phân chia trên đều mang tính phân cấp, nghĩa là có mối quan hệ giữa các thuộc tính, trạm chủ (host) nằm trong mạng (network), phòng (department) nằm trong cơ quan (organization).

Thuộc tính D.vụ/đối tượng	Cấu trúc tên	Phân chia thuộc tính
< thuộc tính>	- cấu trúc phẳng	- vật lý
<tên, thuộc tính, địa chỉ>	- giải pháp cấu trúc phân cấp dựa theo tên (chẳng hạn, trang trắng)	- tổ chức
<tên, kiểu, thuộc tính, địa chỉ>	- giải pháp dựa theo thuộc tính phi cấu trúc	- chức năng

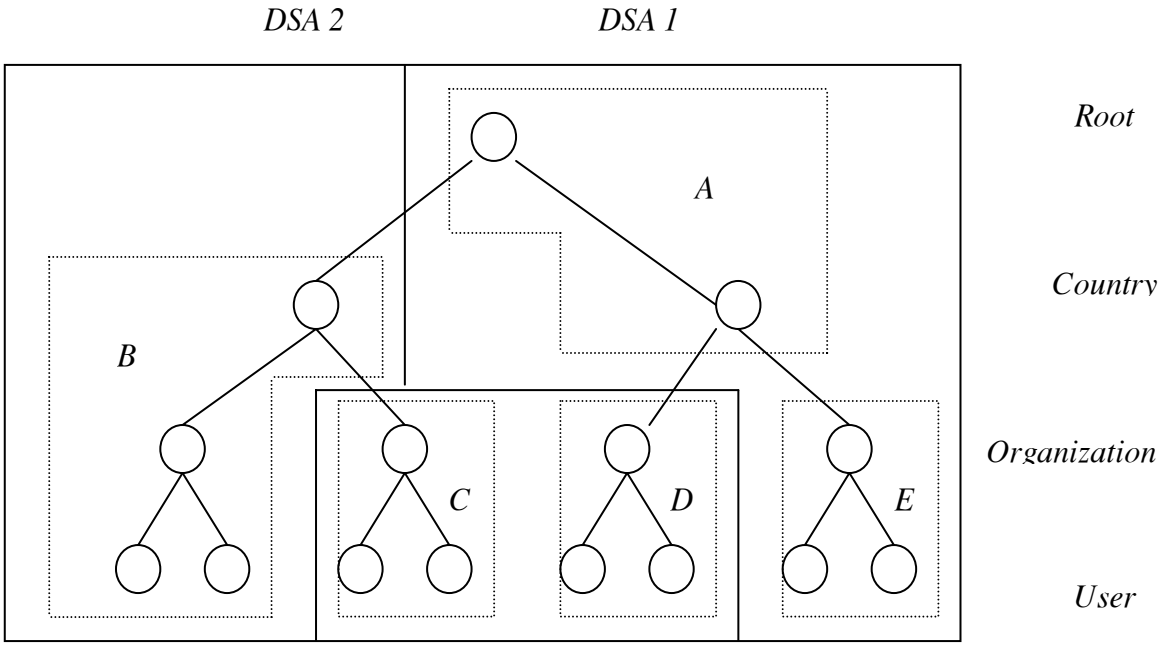
Hình 4.17. Thuộc tính đối tượng và cấu trúc tên

Dùng tên phân cấp rất hiệu quả cho giải pháp tên và quản lý không gian tên. Một số cách đặt tên khác không theo cấu trúc phân cấp chặt chẽ. Ví dụ như thuộc tính nghề nghiệp là <professor>, thuộc tính chuyên môn là <computer science> không được phân chia giống như tên theo tính chất vật lý hoặc tính chất tổ chức của đối tượng. Cách phân chia này được gọi là căn cứ theo chức năng và hợp lý hơn đề xuất giải pháp tên dựa theo thuộc tính phi cấu trúc. Hình 4.17 mô tả các cách khác nhau dùng thuộc tính làm đặc trưng cho mục đối tượng và trình bày cấu trúc tên.

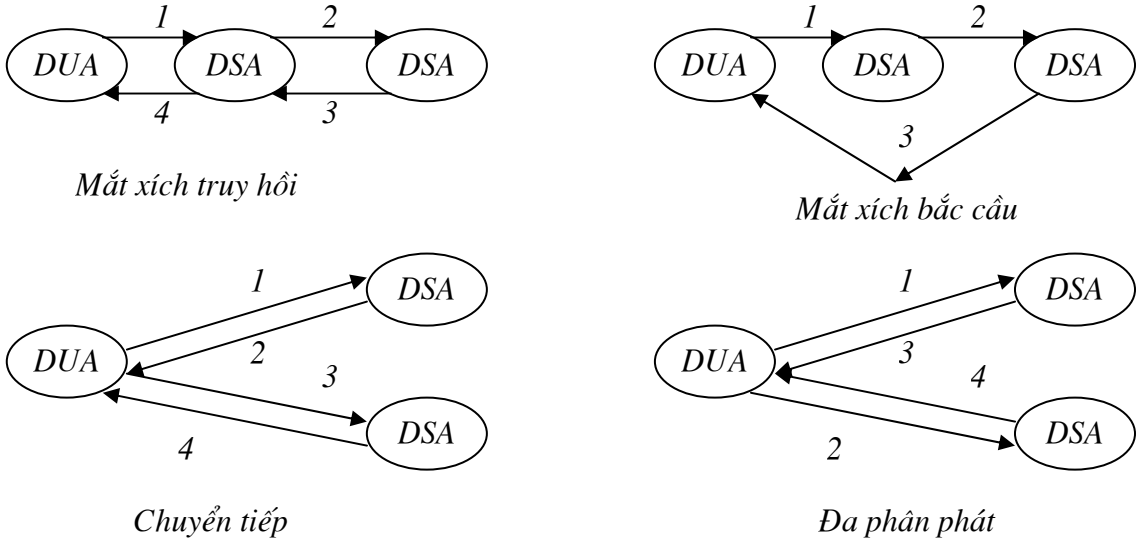
4.4.3. Không gian tên và cơ sở thông tin

Không gian tên và thông tin đối tượng của nó trong hệ phân tán là rất đồ sộ. Để hiệu quả giải pháp và quản lý tên, cần một mô hình thông tin làm cơ sở thi hành cơ sở dữ liệu không gian tên. Theo thuật ngữ X.500, mô hình dữ liệu quan niệm để lưu giữ và trình bày thông tin đối tượng được gọi là cơ sở thông tin thư mục DIB (Directory Information Base). Dịch vụ thư mục (DS) theo chuẩn CCIIT X.500 cung cấp quy tắc cấu trúc và cú pháp đặc tả DIB trong cây thông tin thư mục phân cấp DIT (Directory Information Tree). Hình 4.18 cho ví dụ về DIT. Mỗi mục đối tượng trong DIB được biểu diễn bằng một nút của cây DIT. Các thuộc tính tương ứng với mục đối tượng là tập các thuộc tính từ nút tới gốc cây. Cấu trúc cây có nhiều ưu điểm vì mỗi nút trong cây có đường đi duy nhất tới gốc. Đường đi dùng để định danh duy nhất đối tượng với tên phi cấu trúc trong giải pháp hướng thuộc tính. Thuộc tính dùng cho mục đích đặt tên được gọi là thuộc tính nhận biết (distinguished). Nếu các thuộc tính nhận biết của cùng một kiểu được nhóm lại theo từng mức của cây thì việc ghép nối các thuộc tính nhận biết dọc theo đường đi trở thành tên cấu trúc của đối tượng. Hình 4.18 trình bày cây thông tin thư mục với kiến trúc nhóm các kiểu thuộc tính nhận biết (country, organization, user). Cấu trúc cây phân cấp là hợp lý cho giải pháp tên.

Không gian tên rộng lớn và cây DIT tương ứng của nó có thể được phân tích và phân tán thành miền tên (naming domain) và ngữ cảnh tên (naming context). Miền tên là không gian tên con với một quyền quản trị đơn để quản lý tên. Ngữ cảnh tên là cây con



Hình 4.18. Phân tán của môt DIT



Hình 4.19. Các mode tương tác giải pháp tên

bộ phận của DIT. Hình 4.18 đưa ra 5 ngữ cảnh tên (A, B, C, D, E) trong các vùng có biên rời nét. Ngữ cảnh tên là đơn vị cơ bản để phân tán cơ sở thông tin tới nhà cung cấp dịch vụ thư mục (Directory Service Agent - DSA), là phục vụ cho dịch vụ tên. Có 3 DSA trong vùng với biên liền nét trong hình 4.18 quản lý năm ngữ cảnh tên. Mỗi ngữ cảnh tên cũng tạo nên một cây. Mỗi một DSA duy trì các kết nối tới các DSA khác cộng tác của dịch vụ tên.

Bộ cung cấp thư mục người dùng Directiry User Agent (DUA) thay QT người dùng thực hiện việc khởi tạo QT giải pháp tên. Yêu cầu giải pháp được gửi từ một DSA này tới một DSA khác cho tới khi tìm được đối tượng trong DIT và trả về cho DUA. Mặc dù sơ đồ giải pháp là có cấu trúc-dựa theo tên hoặc phi cấu trúc-dựa theo thuộc tính, tương tác giữa các DSA vẫn có thể thực hiện theo một trong bốn kiểu được trình bày trong hình 4.19.

Trong kiểu mắt xích (chaining mode), yêu cầu được chuyển tiếp từ DSA đến DSA cho tới khi tên được giải quyết. Kết quả được gửi trả lại dọc theo đường cũ (mắt xích truy hồi) hoặc trực tiếp tới DUA (mắt xích ngoài). Mắt xích truy hồi là mode thông dụng đối với giải pháp tên cấu trúc. Kiểu mắt xích ngoài đòi hỏi ít TĐ hơn song mỗi TĐ cần mang địa chỉ của DUA nguồn. Mặt hạn chế chính của mắt xích ngoài là ở chỗ nó không cho phép mô hình lập trình RPC và Client/Server.

Trong kiểu chuyển tiếp (referral mode), khi một DSA không giải quyết được tên yêu cầu từ DUA thì nó gợi ý một DSA khác cho DUA. Kiểu này có thể được dùng cho cả giải pháp tên và giải pháp thư mục.

Trong kiểu đa phân phát (multicast mode), một yêu cầu được định danh và được gửi đồng thời tới các DSA. Một trong những lời đáp có giá trị được DUA chọn. Kiểu này thích hợp khi thông tin cấu trúc là không có sẵn.

Một phương án hợp lý khi kết hợp các mode thao tác khác nhau. Ví dụ, sau bước mắt xích có thể là bước chuyển tiếp.

Tồn tại hai công nghệ chung để cải tiến giải pháp tên là lưu giữ tạm (catching) và nhân bản. Tên được dùng và địa chỉ của chúng được lưu tại một vùng lưu trữ cục bộ để giảm bớt yêu cầu giải pháp tên. Các ngữ cảnh tên cũng được định danh trong các DSA khác nhau để rút ngắn đường giải đáp. Cuối cùng, thực thể đối tượng trong thư mục có thể là một bí danh hay một nhóm. Bí danh và Nhóm là những con trỏ tới tên các đối tượng khác và tới các nút lá trong DIT. Bí danh và Nhóm làm cho dịch vụ tên/thư mục mềm dẻo hơn đối với người sử dụng.

Dịch vụ tên là cốt yếu trong mọi hệ phân tán. Dịch vụ tên trong DCE là ví dụ tốt hỗ trợ lớp rộng lớn các sơ đồ tên. Trong hệ tự trị cộng tác, dịch vụ tên được mở rộng để hỗ trợ hạ tầng truyền thông để định danh và định vị mọi đối tượng tự trị trong hệ thống và quản lý kết nối và phân phát dữ liệu giữa các đối tượng. Bộ môi giới nhu cầu đối tượng - The Object Request Broker (ORB) là một phương tiện trung tâm như vậy trong kiến trúc CORBA đối với hệ tự trị cộng tác.

4.5. Loại trừ ràng buộc phân tán

Chương 3 mô tả ba cách TT chính là một chiều, Client/Server và ngang hàng. Ứng dụng sử dụng TT một chiều thường không cần đồng bộ, TT Client/Server dùng cho nhiều máy khách tạo ra những yêu cầu dịch vụ chia xẻ máy chủ. Nếu đòi hỏi sự cộng tác từ các máy khách thì nó được điều khiển bởi máy chủ và không rõ ràng giữa các QT xử lý của máy khách. Nhưng TTLQT không chỉ giới hạn trong việc tạo ra các yêu cầu dịch vụ. Đôi khi các QT cần trao đổi thông tin với nhau để tạo ra kết luận nào đó về hệ thống hoặc thoả thuận nào đó giữa các QT cùng thực hiện. Những hoạt động này đòi hỏi TT ngang hàng: không có sự chia xẻ đối tượng hoặc tập trung thành phần điều khiển. Hai mục còn lại của chương này trình bày về những vấn đề xảy ra trong cộng tác phân tán khi sử dụng TT ngang hàng.

Nội dung đầu tiên về cộng tác phân tán kiểu phân tán khi xem xét vấn đề đồng bộ loại trừ ràng buộc kinh điển (mục 4.5). Tiếp theo, xem xét một lớp quan trọng cộng tác

phân tán khác dựa trên bầu thủ lĩnh phân tán (mục 4.6). Mục tiêu cơ bản là chỉ ra những khái niệm và vấn đề cơ bản trong cộng tác phân tán.

Loại trừ ràng buộc đảm bảo rằng các QT đồng thời đưa ra các truy nhập tới tài nguyên hoặc dữ liệu chia xẻ (cập nhật CSDL hoặc gửi tín hiệu điều khiển tới thiết bị vào-ra). Thuật toán loại trừ ràng buộc trong hệ phân tán hoạt động theo đúng nghĩa loại trừ nhau (với tính chất tiến bộ nào đó) mà chỉ sử dụng phương thức TT ngang hàng. Thông thường, sử dụng hai cách tiếp cận sau để giải bài toán loại trừ ràng buộc là dựa theo cạnh tranh và dựa theo điều khiển (Thông qua tín hiệu điều khiển - dùng thẻ bài).

Tiếp cận dựa theo cạnh tranh được hiểu là mỗi QT cạnh tranh một cách tự do và bình đẳng để lấy quyền có tài nguyên chia xẻ khi dùng yêu cầu về tiêu chuẩn quyết định. Tiêu chuẩn quyết định có thể dựa trên số lần yêu cầu, tính chất các yêu cầu hoặc biểu quyết.

Trong tiếp cận điều khiển, thẻ lôgic biểu diễn quyền truy nhập tới đối tượng chia xẻ được chuyển theo kiểu quy định giữa các QT cộng tác. QT giữ thẻ được phép đi vào khỏng tới hạn.

Các QT cần cạnh tranh thẻ loại trừ ràng buộc trong thuật toán tiếp cận điều khiển. Trong tiếp cận điều khiển tương tranh, thẻ điều khiển được phân bố theo cách có thứ tự, hiệu quả và tốt đẹp. Có nét tương tự giữa loại trừ ràng buộc phân tán của HĐH với điều khiển truy nhập trung gian của LAN.

4.5.1. Loại trừ ràng buộc theo cạnh tranh

Cạnh tranh vào khoảng tới hạn được quyết định nhờ bất kỳ tiêu chuẩn nhằm tháo bỏ ràng buộc khi các yêu cầu đồng thời xuất hiện. Hợp lý nhất là cấp cho QT đưa ra câu hỏi sớm nhất (theo thời gian logic) hoặc QT nhận được nhiều phiếu bầu nhất từ các QT khác. Gọi hai phương án khác nhau này là các sơ đồ ưu thế tem thời gian và phiếu bầu.

a) Sơ đồ ưu thế tem thời gian

Trong sơ đồ ưu thế tem thời gian, sử dụng khái niệm đồng hồ lôgic làm thứ tự tổng cộng của yêu cầu đi vào khoảng tới hạn. Thuật toán loại trừ ràng buộc phân tán Lamport hoạt động như sau:

(1) QT yêu cầu vào khoảng tới hạn quảng bá REQUEST tới tất cả các QT khác (kể cả nó). Mỗi QT duy trì một dòng đợi các REQUEST chưa giải quyết vào dòng đợi của mình được sắp xếp theo tem thời gian.

(2) Khi nhận được REQUEST, QT lưu TĐ vào hàng đợi yêu cầu của mình và gửi lại REPLY tới QT phát ra yêu cầu.

(3) Khi hàng đợi của QT yêu cầu (phát ra REQUEST) thu thập được N-1 REPLY (lúc này REQUEST đang ở đỉnh hàng đợi) thì QT này được phép vào khoảng tới hạn.

(4) Khi vào khoảng tới hạn, QT gửi thông báo RELEASE tới tất cả các QT và các QT này giải phóng REQUEST khỏi hàng đợi của nó.

Đánh giá thuật toán:

Tổng số thông điệp hoàn thành một lần vào khoảng tới hạn là 3*(N-1) với N là số lượng các QT đang thực hiện.

Cải tiến thuật toán Lamport

Khi quan sát TĐ REPLY bị xóa có thể kết khối QT đi vào khoảng tới hạn, Ricard và Agrawala đã rút gọn độ phức tạp TĐ của thuật toán Lamport. Một số cải tiến như sau:

(1) Nếu QT nhận REQUEST khi nó đang ở trong khoảng tới hạn của nó, hoặc nó đã gửi một REQUEST mà có tem thời gian nhỏ thua tem thời gian của REQUEST đang tới thì nó làm trễ việc phát REPLY.

(2) Chỉ khi QT thu nhận được mọi (N-1) REPLY thì mới được đi vào khoảng tới hạn. Chú ý rằng QT giành dược mọi REPLY chỉ khi nó trở thành QT có ưu tiên cao nhất.

Bản chất là tích hợp hai thông điệp REPLY và RELEASE, số lượng 2*(N-1) TĐ. Thi hành thuật toán ưu thế tem thời gian là đơn giản. Đồng hồ lôgic trong mỗi QT được tăng trưởng theo sự xuất hiện của TĐ thiết lập thứ tự giữa gửi và nhận của các QT và hệ quả là thứ tự tổng cộng của mọi yêu cầu. Thuật toán đạt được loại từ ràng buộc và phát triển bỏ qua sự trì hoãn mập mờ của bất kỳ QU yêu cầu.

b) Sơ đồ phiếu bầu

Theo thuật toán Lamport (hoặc Ricard và Agrawalia), chỉ cần một QT không sẵn sàng là khóa cũng không sẵn sàng. Cần đưa ra sơ đồ mà QT không phải cần giấy phép của tất cả các QT khác để vào khoảng tới hạn. Giống như cuộc đua chính trị, người thắng cuộc có thể được xác định trước khi các phiếu bầu cuối cùng được kiểm.

Có thể áp dụng sơ đồ này cho loại trừ ràng buộc phân tán. QT cần vào khoảng tới hạn được coi là ứng viên. Lá phiếu là TĐ REPLY. QT nào nhận được đa số phiếu thì thắng cuộc, có nghĩa là được phép vào khoảng tới hạn.

(1) Khi nhận được REQUEST, QT gửi REPLY trả lời chỉ khi nó chưa gửi (bầu) cho một ứng viên khác. Mỗi khi QT đã bầu cử, không cho phép nó gửi thêm bất kỳ một REPLY mới cho đến khi phiếu bầu quay về (thông điệp RELEASE).

(2) Ứng cử viên thắng cuộc để đi vào khoảng tới hạn là QT nhận được đa số phiếu bầu. Do chỉ có một ứng viên nhận được đa số phiếu bầu nên loại trừ ràng buộc được đảm bảo.

Có thể xẩy ra vấn đề bế tắc trong sơ đồ phiếu bầu là có ba ứng viên mà mỗi từ chúng nhận được một phần ba số phiếu bầu. Và mường tượng sơ đồ trong đó ứng viên với hầu hết phiếu sẽ là người thắng cuộc. Tuy nhiên, sơ đồ này trở nên phức tạp hơn và tốn kém truyền thông để loại bỏ ràng buộc.

Như giải pháp chọn lựa, một QT có thể thay đổi phiếu bầu của nó khi nhận được yêu cầu từ ứng viên hấp dẫn hơn. Mỗi QT duy trì tem thời gian Lamport và gắn tem đó vào thông điệp REQUEST.

Nếu QT đã bỏ phiếu mà nhận được REQUEST với tem thời gian nhỏ hơn so với tem thời gian của ứng viên mà nó đã bầu thì QT đó lấy lại phiếu bầu đó bằng cách gửi TĐ INQUIRE tới ứng viên. Nếu ứng cử viên này chưa vào khoảng tới hạn thì nó gửi trả lại lá phiếu của người bầu bằng TĐ REINQUISH (ngược lại, khi QT đo đã ở trong khoảng tới hạn thì gửi TĐ RELEASE). Khi QT nhận lại phiếu bầu, nó bỏ phiếu cho ứng cử viên có tem thời gian nhỏ nhất. Kết quả là ứng cử viên nào đó sẽ vào khoảng tới hạn vì chỉ một trong số các ứng cử viên có tem thời gian ngắn nhất.

Đánh giá

Quy tắc ưu thế trong bầu cử đòi hỏi 0 (N) TĐ cho một thực thể khoảng tới hạn (nhiều hơn nếu xảy ra bế tắc). Cải tiến thuật toán nhằm vào rút gọn tổng phí TĐ bằng cách giảm số phiếu cần thiết để vào khoảng tới hạn. Mỗi QT i có tập yêu cầu Si và mỗi QT cần nhận được phiếu bầu từ mọi thành viên trong tập yêu cầu để vào khoảng tới hạn. Nhằm đảm bảo loại trừ ràng buộc, mọi tập yêu cầu phải rời nhau ($Si \cap Sj = null$ với mọi cặp tập yêu cầu khách nhau). Các tập yêu cầu hợp lý cho loại trừ ràng buộc thường

được coi là tập quy định (quorum), và tập các tập yêu cầu được coi là phái (coterie). Còn có các điều kiện khác cho tập quy định và phái không quan hệ trực tiếp với tính chính xác song đáng được thi hành. Nếu không có một tập quy định nào là tập con thực sự của một tập quy định khác thì phái là tối thiểu. Nếu các tập quy định cùng kích thước thì phái đòi hỏi sự cố gắng như nhau và trách nhiệm như nhau.

Trong tình huống với tập điều kiện cần có đã cho, bài toán xác định phái đảm bảo các tính chất mỗi tập quy định có cùng kích thước tối thiểu. Hợp lý mỗi tập quy định có cỡ $O(\sqrt{N})$.

4.5.2. Loai trừ ràng buộc dưa vào thẻ bài

Mặc dù các thuật toán loại trừ ràng buộc phân tán dựa theo cạnh tranh tuy có tính hấp dẫn song tổng phí TĐ lại cao. Một lựa chọn thuật toán dựa theo cạnh tranh khác là dùng thẻ bài điều khiển hiển, sở hữu nó được quyền vào khoảng tới hạn. Nếu chỉ có một thẻ bài, loại trừ ràng buộc được đảm bảo. Các phương pháp khác nhau về yêu cầu và chuyển thẻ bài cho hiệu năng và tính tốt đẹp khác nhau. Nhằm đảm bảo rằng yêu cầu đi tới QT giữ thẻ và thẻ đi tới được QT yêu cầu, thuật toán đòi hỏi một cấu trúc lôgic của tập các QT. Ba cấu trúc thường gặp là cây (Tree), vòng (Ring) và quảng bá (Broadcast).

Cấu trúc vòng (Ring structure)

Các QT được nối theo một vòng logic và một thẻ bài di chuyển trên vòng. QT sở hữu thẻ bài nó được phép vào khoảng tới hạn. Khi kết thúc khoảng tới hạn (nếu có), QT chuyển thẻ bài tới QT kế tiếp trong vòng. Cấu trúc vòng đơn giản, dễ thực hiện và tránh được bế tắc. Tuy nhiên, thẻ bài cần lưu chuyển trong vòng thậm chí không có QT nào muốn vào khoảng tới hạn của nó tạo ra giao thông mạng không cần thiết. Hơn nữa, một QT muốn vào khoảng tới hạn buộc phải chờ cho tới khi thẻ bài xuất hiện. Việc chờ đợi này có khi rất lớn, ngay cả khi không có một QT khác muốn vào khoảng tới hạn, vì rằng thẻ bài buộc phải di chuyển theo vùng lớn của vòng mới tới được QT yêu cầu. Khi số lượng yêu cầu vào khoảng tới hạn cao, thuật toán vòng lôgic làm việc tốt vì thẻ bài chỉ phải chuyển rất ít bước trong vòng để QT vào khoảng tới hạn.

Cải tiến đáng ghi nhận của tiếp cận dựa theo thẻ bài so với tiếp cận dựa theo cạnh tranh là thẻ bài được dùng mang theo trạng thái. Ví dụ, một sơ đồ ưu thế có thể thi hành bằng việc gắn thông tin ưu tiên vào thẻ bài. QT xác nhận thẻ bài chỉ khi nó nhận được thẻ bài và độ ưu tiên của nó cao hơn độ ưu tiên gắn trên thẻ bài. Độ ưu tiên có thể được người dùng xác định hoặc dùng tem thời gian.

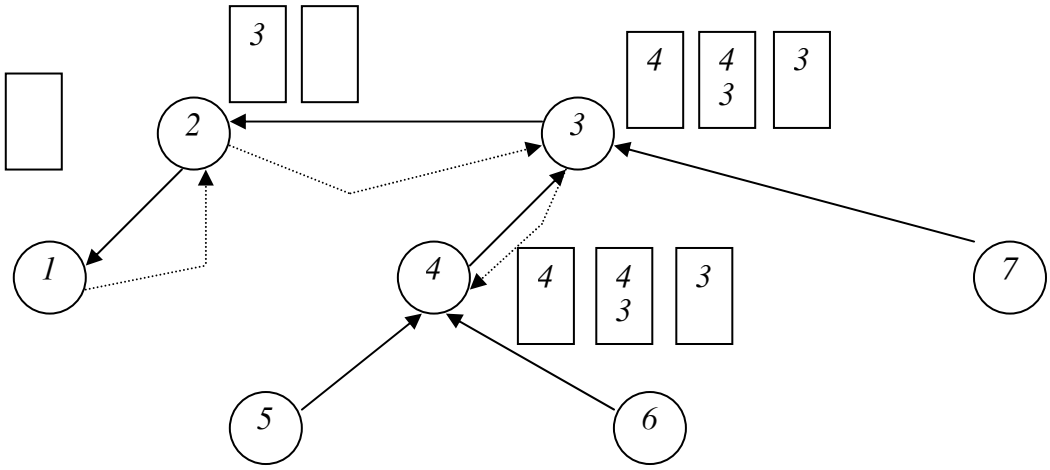
Các chuẩn IEEE 802.4 (Token Bus) và IEEE 802.5 (Token Ring) đối với LAN sử dụng phương pháp này. Các giao thức LAN chuẩn này đặc tả các thủ tục để cấu hình vòng, kiểm soát độ ưu tiên và lỗi hệ thống. Sự khác nhau cơ bản là là giả thiết về kiến trúc mạng. Token Bus và Token Ring dùng kiến trúc kiến trúc tuyến và vòng phần cứng, trong khi quan tâm loại trừ ràng buộc ở dây là bài toán mức ứng dụng và không có một giả thiết nào về mạng hạ tầng. Chức năng giao thức đòi hỏi năng lực chẳng hạn giám sát tuyến trong Token Bus không thể được thi hành trong mức ứng dụng. Tuy nhiên, dùng thẻ mang thông tin giữa các nút cộng tác có thể tạo thuận lợi cho cộng tác phân tán.

Cấu trúc cây (Tree Structure)

Vấn đề nảy sinh trong cấu trúc vòng là thẻ bài rỗi (Thẻ bài không được sử dụng) cứ di chuyển mãi trên vòng khi không có QT nào cần đến nó. Một lựa chọn khác là QT cần khẳng định rõ yêu cầu thẻ bài và chỉ di chuyển thẻ bài khi biết có yêu cầu chưa quyết

định. Do yêu cầu buộc phải tìm được thẻ bài, có thể dẫn tới tình huống trì hoãn và bế tắc không rõ ràng. Có thể thấy rằng cấu trúc vòng không là tốt nhất để đạt được thẻ bài vì đường đi dài (cho yêu cầu hoặc cho thẻ bài), mà trường hợp tồi nhất cần n-1 đoạn (n là số nút trong vòng). Để giải quyết vấn đề này, thuật toán Raymond sử dụng cấu trúc cây lôgic (Hình 4.20).

Trong cây logic, thẻ bài luôn thường trực tại gốc cây. Chú ý là mũi tên về gốc cây, là ngược với quy ước thông thường. Mỗi nút cây thể hiện vị trí logic hiện tại của một QT.



Hình 4.20. Truyền thẻ bài cấu trúc cây

Khi một QT yêu cầu thẻ bài, nó gửi yêu cầu theo đường đi tới gốc. Nếu thành công, thẻ bài sẽ chuyển tới nút đó và tạo thành một cây logic mới với gốc là nút nhận thẻ bài. Ở đây chỉ trình bày những nội dung cơ bản nhất của thuật toán Raymond.

Mỗi QT duy trình một dòng đợi FIFO các yêu cầu và hướng tới QT tiền nhiệm trực tiếp (đích của cung xuất phát từ QT đang xét). Khi một QT nhận được một yêu cầu, nó bổ sung yêu cầu đó vào cuối của dòng đợi FIFO. Nếu dòng đợi rỗng và nó không có thẻ bài thì QT cần thực hiện thao tác chiếm giữ thẻ bài thông qua việc nó yêu cầu thẻ bài từ QT tiền nhiệm. Ngược lại, QT hoặc đã có thẻ bài hoặc sẽ sớm nhận được thẻ bài, nó không đưa ra một thao tác nào. QT yêu cầu thẻ bài được sinh ra theo yêu cầu cục bộ mà biểu diễn như tiến trình trên.

Khi QT có thẻ bài mà không sử dụng nó và có dòng đợi FIFO khác rỗng, nó tháo bỏ thực thể QT đầu tiên từ dòng đợi FIFO và gửi thẻ bài tới QT này. Điều kiện khởi động xuất hiện khi một yêu cầu đi tới, khi thẻ bài đi tới hoặc QT giải phóng thẻ bài. Do QT không giữ thẻ bài lâu nên nó thay đổi con trỏ tới QT mà nó sẽ gửi thẻ bài tới. Nếu dòng đợi FIFO khác rỗng, QT cần giành lại thẻ bài, vì vậy nó gửi yêu cầu tới QT mới nắm giữ thẻ bài. Một loại bỏ xuất hiện nếu QT là thực thể đầu tiên trong dòng đợi FIFO. Trong trường hợp này, QT đang trong khoảng tới hạn.

Hình 4.20 chỉ dẫn cách cấu trúc cây thay đổi động. Thẻ bài được khởi động tại nút 1. Nút 4 muốn chiếm thẻ bài sớm nhất, đưa ra thao tác yêu cầu thẻ bài tới nút 3. Theo đó, nút 3 chuyển tiếp yêu cầu từ nút 4 tới nút 2. Thẻ bài di chú từ nút 1 tới nút 4 thông qua các nút 2 và 3. Nút 3 chuyển thẻ bài và gửi yêu cầu tới nút 4 do dòng dợi tại nút 3 khác rỗng (nút 3 yêu cầu thẻ bài sau nút 4). Đường vẽ trong hình là các thay đổi chỉ dẫn di chú thẻ bài và số trong các hộp cho biết nội dung của các yêu cầu theo các bước.

Thuật toán là thi hành phân tán dòng đợi FIFO toàn cục. Dòng đợi FIFO cục bộ liên kết với FIFO toàn cục sử dụng kiến trúc cây lôgic. Tính phi chu trình của cấu trúc cây làm cho nó đạt được cái không thể đạt được từ danh sách đợi vòng, loại bỏ khả năng bế tắc. Mỗi nút biểu diễn cây con gửi yêu cầu đơn tới nút tiền nhiệm, đưa yêu cầu thẻ bài đối với thực thể cây con. Dòng yêu cầu tại mỗi nút cho thứ tự ưu tiên các yêu cầu từ cây con người thắng cuộc theo thứ tự FIFO đảm bảo tính tốt và tính tự do khỏi sự trì hoãn mập mờ. Cây lôgic có thể thay đổi kiến trúc lôgic của nó nhằm làm tăng hệiu lực bản chất. Một số giải pháp cải tiến đã được đề xuất.

Cấu trúc quảng bá

Sử dụng kiến trúc lôgic làm tăng tính hiệu quả song việc thi hành thuật toán lại phức tạp hơn do phải khởi tạo và duy trì kiến trúc đó. Trong suốt hơn và là điều mong muốn cho truyền thông nhóm là không cần nhận thức về kiến trúc. Các thuật toán loại trừ ràng buộc dựa theo cạnh tranh trước đây dùng quảng bá và giải quyết cạnh tranh bằng tem thời gian hoặc phiếu bầu. Nếu sử dụng cách quảng bá để cạnh tranh thẻ bài, thì bản chất vẫn là thuật toán dựa theo cạnh tranh. Tuy nhiên, đây là một cải tiến đáng kể dùng thẻ bài. Thẻ bài có thể mang thông tin toàn cục hữu dụng cho cộng tác QT. Thẻ bài điều khiển cho loại trừ ràng buộc được tập trung hóa và được dùng để xếp hàng các yêu cầu tới khoảng tới hạn. Đó là những nội dung cơ bản nhất của thuật toán quảng bá Suzuki/Kasami.

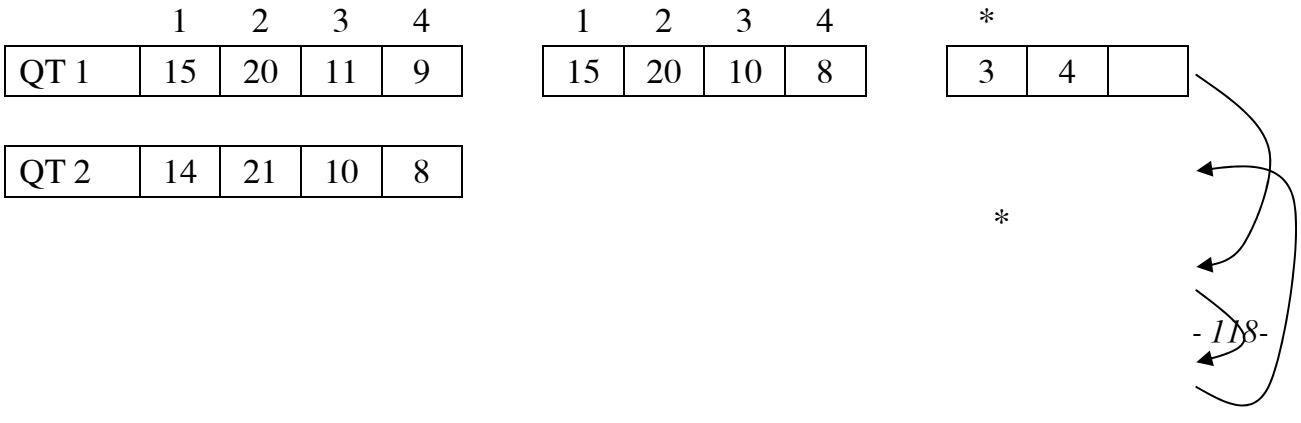
Thẻ bài điều khiển tương ứng với một cấu trúc dữ liệu chứa một vector thẻ bài T và một dòng xếp hàng các yêu cầu Q. Thực thể trong T được chỉ số hóa bằng số hiệu QT và trình bày số tích luỹ việc hoàn thành khoảng tới hạn của QT tương ứng. Mỗi QT giữ một số hiệu dãy cục bộ, chính là số lần QT đã đòi hỏi vào khoảng tới hạn. Số hiệu dãy này được gắn vào mọi TĐ REQUEST mà QT này quảng bá. Mọi QT p còn duy trì một vector dãy S_p, chứa chỉ số dãy cao nhất của mọi QT mà p chấp nhận. Dòng đợi yêu cầu trong thẻ bài là danh sách các yêu cầu theo thứ tự FIFO. Hoạt động của thuật toán được trình bày như dưới đây.

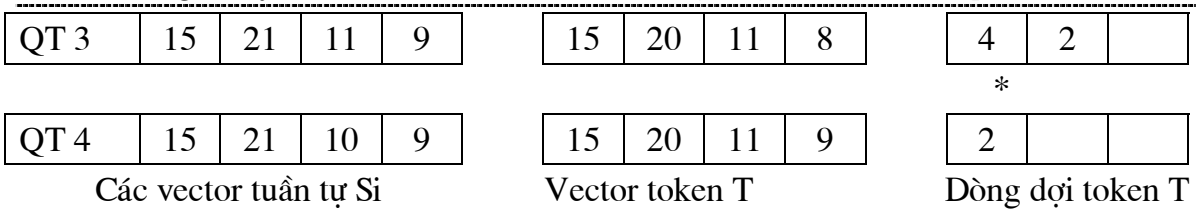
Thuật toán Suzuki/Kasami

(1) Để yêu cầu vào khoảng tới hạn, QT i quảng bá TĐ REQUEST kèm theo số hiệu dãy đã được tăng seq = $S_i[i]$ tới tất cả các QT trong nhóm,

(2) Mọi QT j khi nhận được TĐ REQUEST từ QT i, cập nhật vector dãy qua tính toán $S_j[i] = \max (S_j[i], seq)$. Nếu $S_j[i]=T[i]+1$ và QT j đang mang thẻ bài rỗi (tức hàng đợi Q rỗng) thì nó gửi thẻ bài tới i.

(3) QT i nhận được thẻ bài và vào khoảng tới hạn. Dòng đợi yêu cầu trong thẻ bài có Q khác rỗng nếu thẻ bài được chuyển từ một QT đã biển đổi Q. Vào lúc hoàn thành khoảng tới hạn, QT i cập nhật vector thẻ bài bằng cách đặt $T[i]=S_i[i]$ và so sánh S_i với T để bổ sung vào Q mọi QT có $S_i[k]=T[k]+1$ với $k \neq i$ nếu chúng chưa sẵn trong dòng đợi yêu cầu. Sau khi cập nhật Q, QT i loại đỉnh của dòng đợi yêu cầu và gửi thẻ bài tới QT trên đỉnh (sau khi loại đỉnh). Nếu Q rỗng, thẻ bài ở lại với i.





Hình 4.21. Thuật toán quảng bá dựa theo thẻ bài

Ví dụ được minh họa thuật toán cho trong hình 4.21 với bốn QT cộng tác, vector dãy là (14, 20, 10, 8). Ban đầu QT 1 đang giữ thẻ bài và vào khoảng tới hạn, ba QT kia yêu cầu thẻ bài. Do sự trễ của mạng giữa QT1 và QT 2, giả sử 3 và 4 gửi yêu cầu tới QT 1 trước. QT 1 chấp nhận yêu cầu từ 3 và 4. Nó bổ sung các yêu cầu này vào hàng đợi và gửi thẻ bài tới QT 3. Thực thể đỉnh của dòng đợi yêu cầu (vị trí dấu *, là số 3) bị loại khỏi dòng đợi yêu cầu trước khi thẻ bài được gửi tới QT 3. Sau khi QT 3 kết thúc, nó bổ sung yêu cầu của QT 2 vào hàng đợi thẻ bài và gửi thẻ bài tới QT 4. Cuối cùng, QT 4 chuyển tiếp thẻ bài cho QT 2 và thẻ bài nằm lại ở đây do không còn yêu cầu thẻ bài.

Thuật toán quảng bá thẻ bài là đơn giản và hiệu quả, đặc biệt với hệ thống có phương tiện quảng bá hiệu quả. Tuy thuật toán Suzuki/Kasami không hoàn toàn phân tán như tiếp cận dựa theo cạnh tranh, song ở dây cũng không có điều khiển tập trung và quản lý thẻ bài chia xẻ là phân tán. Chỉ có cạnh tranh loại trừ ràng buộc là thi hành tập trung bởi dòng đợi FIFO thẻ bài. Thuật toán không bị bế tắc hoặc bỏ sót.

4.6. Bầu thủ lĩnh

Sử dụng bộ điều khiển tập trung làm cho đồng bộ QT trở nên đơn giản. Tuy nhiên, bộ điều khiển tập trung lại là tâm điểm của lỗi và làm hạn chế hiệu lực của dịch vụ. Vấn đề được giảm nhẹ nếu bộ phối hợp (bộ điều khiển - thủ lĩnh) mới được chọn chống lại lỗi của bộ phối hợp hiện thời. Bầu thủ lĩnh liên quan tới việc bầu một QT thủ lĩnh duy nhất, được mọi QT khác trong nhóm biết. Việc bầu thủ lĩnh xuất hiện trong lúc khởi tạo hệ thống hoặc khi thủ lĩnh hiện thời bị hỏng. Quá trình bầu thủ lĩnh mới được kích hoạt khi lỗi được phát hiện hoặc có nghi ngờ. Khám phá lỗi thông thường dựa vào quá hạn. Một QT không nhận được trả lời từ thủ lĩnh trong ngưỡng quá hạn được xác định trước đưa đến việc nghi ngờ thủ lĩnh bị hỏng và khởi tạo quá trình bầu thủ lĩnh. Chú ý rằng báo động sai có thể xuất hiện, và thuật toán bầu thủ lĩnh phải biết được tình huống này. Các báo động sai sẽ hiếm nếu ngưỡng quá hạn được chọn thích hợp.

Trong thuật toán đồng bộ theo thẻ bài, QT giữ thẻ bài có thể được coi là thủ lĩnh của hệ thống. Trong trường hợp này, vai trò thủ lĩnh được luân phiên giữa các QT và QT không cần biết định danh của thủ lĩnh hiện thời. Mất thẻ bài tương đương lỗi của thủ lĩnh.

Tồn tại hai chiến lược bầu thủ lĩnh: (1) bầu thủ lĩnh dựa trên độ ưu tiên toàn thể (*global priority)*. Kiểu này được gọi là tìm kiếm cực trị (*extrrma finding)***,** (2) các QT trong nhóm "bầu" thủ lĩnh dựa trên những độ ưa thích riêng tư (vị trí, độ tin cậy vv..) Lớp sơ đồ phiếu bầu được gọi là thuật toán bầu thủ lĩnh dựa theo ưa thích (***preference-base***). Chiến lược phiếu bầu (2) là phổ dụng hơn chiến lược tìm kiếm cực trị (1). Ví dụ, một ứng cử viên mạnh hơn người khác hoặc nhận được nhiều phiếu bầu hơn là kết quả của quyết định phức tạp hơn và hậu quả dự đoán được ít hơn.

Theo nhiều khía cạnh, bầu thủ lĩnh và loại trừ ràng buộc phân tán là như nhau, cả hai đều cố gắng đạt tới sự đồng thuận định danh một QT duy nhất. Tuy nhiên, tồn tại một số khác biệt thú vị. Trong bầu thủ lĩnh, QT có thể chịu thua một QT khác để trở về trạng thái hoạt động bình thường của mình cho đến khi thủ lĩnh được lựa chọn, còn

trong loại trừ ràng buộc phân tán, QT cạnh tranh cho đến khi nó thành công. Thuật toán loại trừ ràng buộc phải đảm bảo rằng không có QT nào bị lãng quên trong khi thuật toán bầu thủ lĩnh lại cố gắng đẩy nhanh và kết thúc kết quả quá trình bầu cử. Nét đặc biệt nữa là kết quả của bầu thủ lĩnh phải được thông báo cho tất cả các QT khác. Với loại trừ ràng buộc, nếu chưa tham gia vào khoảng tới hạn thì QT không quan tâm bất cứ QT nào hiện đang ở khoảng tới hạn.

Giống như thuật toán loại trừ phân tán, thiết kế của thuật toán bầu thủ lĩnh cũng phụ thuộc vào giả thiết cấu trúc kiến trúc của nhóm QT. Ba loại kiến trúc thông dùng là *đầy đủ, vòng và cây* được trình bày trong phần tiếp theo. Trong thuật toán, hai QT khác nhau có độ ưu tiên khác nhau.

4.6.1. Kiến trúc đầy đủ

Trong kiến trúc đầy đủ, mỗi QT trong nhóm tham chiếu tới các QT khác trong cùng nhóm chỉ cần gửi một TĐ. TĐ bầu cử được gửi theo chế độ điểm-điểm từ QT này đến QT khác. Giả thiết đầu tiên trong môi trường HĐH là tất cả chỉ số các QT phải duy nhất và mọi QT khác đều biết. Do tiến trình bầu thủ lĩnh được khởi xướng bằng phát hiện lỗi, giả thiết tiếp theo là về mô hình lỗi và cơ chế phát hiện lỗi. Để đơn giản, giả sử có mô hình lỗi ngây thơ. Giả thiết thứ hai là mạng truyền thông là tin cậy và chỉ có QT truyền thông có thể bị lỗi. Hệ mạng TT tin cậy có nghĩa là TĐ truyền đi không bao giờ bị thất lạc, thay đổi, bị trùng lặp và được phân phát đúng thứ tự trong thời gian hữu hạn đã biết. Giả thiết thứ ba liên quan đến lỗi của QT. Một QT nắm giữ TĐ trong một khoảng thời gian hữu hạn đã biết. Lỗi QT ngừng tính toán và không sinh ra TĐ báo lỗi làm hệ thống lẫn lộn. Hơn nữa, khi QT khôi phục, nó có kinh nghiệm về lỗi. Giả thiết thứ hai và thứ ba tạo hợp lý để nhận biết lỗi và gắn lại nút lỗi vào nhóm. Lỗi được phát hiện tin cậy bằng khởi tạo một khoảng quá hạn sẽ làm tăng chút ít tổng độ trễ TĐ khứ hồi và thời gian xử lý TĐ. QT bị lỗi được gắn vào nhóm bằng bầu cử bắt buộc trong khi khôi phục QT. Như một lựa chọn, QT bị lỗi tin tưởng việc cộng tác bằng việc bầu cử định kỳ đối với các QT được khôi phục sẽ được gắn với nhóm. Với những giả thiết trên đây, Garcia-Molina đầ xuất thuật toán bầu thủ lĩnh điển hình, được gọil à thuật toán kẻ mạnh (bully).

Thuật toán Bully là thuật toán tìm kiếm cực trị. Mỗi QT có độ ưu tiên toàn cục (có thể dùng chỉ số QT), QT nào có độ ưu tiên cao nhất là thủ lĩnh được bầu. Một QT bắt đầu bầu thủ lĩnh nếu nó nghi ngờ bộ phối hợp bị sự cố (ví dụ như, phát hiện bằng ngưỡng quá hạn) hoặc khi nó khôi phục sau sự cố. QT bắt đầu bằng việc gửi một TĐ dò hỏi *are-you-up* tới mọi nút có độ ưu tiên cao hơn để kiểm tra sự tồn tại của các QT đó.

Nếu ít nhất có một TĐ trả lời cho TĐ dò hỏi trên thì QT tự từ bỏ cố gắng ứng cử và chờ nút có độ ưu tiên cao hơn ứng cử.

Nếu không có TĐ trả lời nào từ các nút có độ ưu tiên cao hơn thì QT tự đặt mình làm thủ lĩnh bằng cách gửi TĐ yêu cầu *enter-election* tới mọi nút có độ ưu tiên thấp hơn. TĐ *enter-election* báo cho các QT có độ ưu tiên thấp hơn biết là tiến trình bầu cử đang xảy ra và các QT này nên chuẩn bị chấp nhận và giao tiếp với nó theo vai trò thủ lĩnh mới. Bầu cử chỉ hoàn thiện cho đến khi kết quả được gửi và nhận từ mọi nút đang hoạt động có độ ưu tiên thấp hơn.

Tại cùng thời điểm, có thể một vài QT cùng tự ứng cử vai trò thủ lĩnh. Chẳng hạn, QT có độ ưu tiên cao bị lỗi được khôi phục vào thời điểm đang diễn ra một cuộc bầu cử. Để đảm bảo tính nhất quán, QT chuyển vào trạng thái tạm thời khi nhận TĐ *enter-election*. QT ứng cử tự khai báo trở thành thủ lĩnh mới chỉ sau khi đã nhận được trả lời của mọi TĐ *enter-election* hoặc quá hạn thời gian trả lời từ mọi QT có độ ưu tiên thấp hơn. Lúc này mọi QT hoặc bị lỗi, thực hiện thủ tục khôi phục, hoặc đang ở trạng thái

tạm thời. Điều này đảm bảo QT khởi tạo khai báo tự là thủ lĩnh, do không còn QT nào khác nghĩ rằng còn có một thủ lĩnh khác.

Cuối cùng, QT khởi tạo phân bố trạng thái mới và thông báo cho mọi QT khác biết nó là thủ lĩnh trong tính toán mới.

4.6.2. Kiến trúc vòng logic

Thuật toán bầu thủ lĩnh sẽ hết sức đơn giản nếu cho trước kiến trúc cố định kết nối mọi QT cộng tác. Do kiến trúc chỉ được dùng để thuận tiện truyền thông, giả thiết kiến trúc đơn giản nhất là kiến trúc vòng logic.

Vòng logic là dễ dàng xây dựng và cho một thuộc tính có một không hai là TĐ được gửi từ bất kì nút nào cũng có thể trở về nút đó, biểu diễn đủ một vòng thao tác hoặc quảng bá mà không cần thêm tri thức.

Để tìm QT có độ ưu tiên cao nhất trong vòng logic, QT khởi tạo bắt đầu tuần hoàn TĐ nhờ gắn vào TĐ độ ưu tiên (hoặc định danh **id**) của mỗi nút dọc theo vòng. Khi TĐ quay trở lại nút khởi tạo, nó chọn QT có độ ưu tiên cao nhất trong TĐ và quảng bá định danh thủ lĩnh mới tới mọi nút, một vòng nữa gọc theo vòng. Thuật toán có thể được cải tiến theo hai hướng. Thứ nhất, không cần thiết phải thu thập mọi định danh vào TĐ đơn. Để tìm định danh lớn nhất, mỗi nút đơn giản chỉ chuyển tiếp giá trị lớn hơn giữa định danh của nó và giá trị nhận được trên đường truyền tới nút tiếp theo. Theo cách này, định danh cực đại được được nổi lên dọc theo vòng và cuối cùng thu được nút thủ lĩnh. Thứ hai, nút rơi vào tiến trình bầu cử không cần quan tâm đến TĐ ngoại trừ TĐ chứa giá trị cao hơn so với định danh của bản thân nó.

Thuật toán vòng lôgic cải tiến do Chang và Roberts phát triển, được trình bày trong hình 4.22.

```
Nút khởi tạo khởi động participant = true và send (id) tới nút tiếp theo ;
For mỗi nút_QT,
    receive(value);
    case
        value > id: participant := true, send (value);
        value < id and participant == false: participant := true, send (id);
        value = id: công bố thủ lĩnh
    end case
```

Hình 4.22. Thuật toán bầu cử vòng của Chang và Robert

Mỗi QT đều có biến cục bộ là participant (được khởi tạo là false) cho biết QT này đã tham gia vào tiến trình bầu cử hay chưa. QT khởi tạo dị bộ bắt đầu bầu cử bằng việc gửi TĐ chứa định danh của nó. Mỗi QT nhận TĐ so sánh định danh của nó với giá trị tại TĐ. Giá trị lớn hơn được chuyển tiếp tới nút tiếp theo và QT đặt giá trị cho biến participant=True**.**

Một QT đã tham gia tiến trình bầu cử, nhận được TĐ bầu cử có giá trị định danh nhỏ hơn của nó thì nó không chuyển tiếp TĐ nữa. Thủ lĩnh tìm được khi TĐ mang giá trị cực đại đi đủ một vòng và trở về nút nút có định danh lớn nhất. Khi đó, thủ lĩnh gửi định danh của nó lên toàn bộ vòng, thông báo tới mọi nút về bộ phối hợp mới và đăth giá trị biến participant của các nút đó là False**.**

Tình huống bình thường, khi chỉ có một bộ khởi tạo, thuật toán của Chang và Robert mất trung bình N/2 TĐ để đạt được nút định danh lớn nhất và mất khoảng N TĐ khác để quay trở về. Như vậy, độ phức tạp thời gian và TĐ của thuật toán là O(N). Độ phức tạp cao hơn trong trường hợp đặc biệt khi mọi nút đều khởi phát việc bầu cử cùng lúc. Nếu không tối ưu, N khởi tạo bầu cử chiếm mỗi cái 0(N) TĐ, và tổng cộng có $0(N^2)$. được gửi. Như vậy, trường hợp tốt nhất và tồi nhất của thuật toán vòng lôgic tương ứng là **O(N)** và **$O(N^2)$**, phụ thuộc vào định danh các nút trên vòng đã được xếp tăng dần hoặc giảm dần.

Một số thuật toán bầu thủ lĩnh vòng với độ phức tạp TĐ trong khoảng **O(NlogN)** đã được đề xuất. Tư tưởng của các thuật toán này là giảm khởi tạo bầu cứ từ nút độ ưu tiên thấp càng nhiều càng tốt, bất luận thứ tự kiến trúc của các nút. Điều này thực hiện bằng cách so sánh định danh một nút với định danh hai nút kề cận của nó. Nút khởi tạo ở trạng thái chủ động nếu có định danh lớn hơn định danh của hai nút kề cận. Ngược lại nó trở thành bị động và chỉ được quyền nhận TĐ. Cách này loại bỏ hiệu quả ít nhất một nửa các nút mỗi vòng chuyển TĐ và kết quả chỉ còn **log(N).**

Hướng tiếp cận này đòi hỏi vòng hai chiều. Với vòng một chiều, có hiệu quả tương đương với so sánh ba nút khi dùng bộ đệm lưu hai TĐ liên tiếp, trước khi một nút được xác định là chủ động hoặc bị động.

4.6.3. Kiến trúc cây

Trong thuật toán bầu thủ lĩnh theo vòng logic, chưa tính đến vấn đề quản lí vòng logic. Nói riêng, xây dựng vòng như thế nào và làm cách nào duy trì hình trạng vòng khi đối phó với sự cố nút. Xây dựng và quản lí kiến trúc vòng logic là dễ dàng hơn nếu hạ tầng mạng có các kênh truyền thông chia xẻ, sẵn có các phương tiện phần cứng quảng bá. Vòng lỗi được phát hiện và cấu hình lại một cách hiệu quả nhờ việc giám sát và quảng bá TĐ trên kênh. Trong các mạng phổ dụng hớn với với kiến trúc mạng bất quy tắc, quảng bá được mô phỏng bằng đơn phát điểm-điểm phức. Một số phương pháp xây dựng vòng logic trong mạng bất quy tắc được đề xuất. Ích lợi của việc có kiến trúc lôgic là rõ ràng. Nó xác định và kết nối nhóm QT, thuận tiện trong việc bầu thủ lĩnh và được dùng một cách hiệu quả để truyền thông giữa thủ lĩnh và các nút con.

Cây bao trùm được dùng như cấu trúc kiến trúc biểu diễn việc việc liên kết giữa các nút thành viên trong nhóm QT. Mạng có **N** nút được biểu diễn bằng một đồ thị với **E** cung nối các nút. Mỗi nút được coi là một thực thể tự trị có trao đổi TĐ với các nút kề cận nó. Giá truyền thông từ một nút tới nút kề cận của nó được biểu thị bằng trọng số của cung ra và được nút đó biết. Cây bao trùm cấp **N** là một cây biểu diễn mạng **N** nút với **N-1** cạnh. Cây bao trùm tối thiểu *(MST)* là cây bao trùm với tổng trọng số các cung là nhỏ nhất. Cây bao trùm là một đồ thị phi chu trình. Nút bất kỳ của cây đều có thể được coi là gốc, hay cũng thế là thủ lĩnh của cây. Mỗi nút có duy nhất một đường tới gốc để đưa ra yêu cầu tới thủ lĩnh. Hệ quả là, cấu trúc cây cũng được dùng trong việc bầu cử thủ lĩnh mới. Thuật toán phân tán xây dựng cây bao trùm tối thiểu trong một mạng là một trong những vấn đề nghiên cứu mạnh mẽ trong tính toán phân tán. Bài toán bầu thủ lĩnh và cũng như xây dựng cây bao trùm tối thiểu dễ dàng được thu gọn về nhau. Một thuật toán phân tán hiệu quả giải bài toán này được chuyển đổi thành thuật toán giả bài toán kia với chỉ những biến đổi nhỏ.

Gallager, Humbelt và Spira tiên phong trong việc đề xuất thuật toán tìm cây bao trùm tối thiểu phân tán. Thuật toán của họ dựa trên các bước *tìm kiếm* và *kết hợp*. Thuật toán hợp nhất các đoạn, đi ra từ mỗi nút hiện là đoạn mức 0. Các đoạn đó được hợp nhất lại từng mức theo thuật toán Bottom-up cho đến khi được đoạn kết quả bao gồm các cung của cây bao trùm tối thiểu. Các đoạn là các cây con trọng số tối thiểu trong cây bao

trùm tối thiểu. Mỗi đoạn, một cách dị bộ và độc lập, tìm cung ra với trọng số nhỏ nhất của nó và dùng cung này để nối với một nút trong một đoạn khác để hình thành một đoạn lớn hơn của MST. Cây thu được nhờ hợp nhất hai cây con có trọng số nhỏ nhất khi sử dụng cạnh có trọng số nhỏ nhất cho kết quả là cây có trọng số nhỏ nhất.

Theo mục đích bầu thủ lĩnh, bất cứ cây bao trùm nào cũng đều đáp ứng được mọi nhu cầu. Trọng số các cung không mang theo trong việc tìm thủ lĩnh tốt nhất, vì vậy cây có giá tối thiểu hay không là không liên quan. Tuy nhiên thuật toán là phân tán và vì vậy cây bao trùm cuối cùng buộc là duy nhất phù hợp khi kết thúc thuật toán. Nếu không, thuật toán có thể không kết thúc hoặc có thể bị bế tắc. Đây là lý do phải tìm một cây bao trùm tối thiểu và thuật toán hoạt động được chỉ khi cây bao trùm tối thiểu là duy nhất. Điều này biểu thị rằng mọi cung trong đồ thị liên thông có trọng số duy nhất và cây MST tồn tại duy nhất. Với bài toán bầu thủ lĩnh, giả thiết rằng cung ra từ một nút được gắn nhãn duy nhất và gắn kết các định danh nút. Nếu định danh nút là duy nhất trong toàn bộ hệ thống thì trọng số cung cũng duy nhất.

Thủ lĩnh được chỉ định như nút cuối cùng được hợp nhất khi sinh ra cây bao trùm tối thiểu. Có thể bầu thủ lĩnh sau khi cây bao trùm tối thiểu đã được xây dựng. Nút khởi tạo quảng bá TĐ Campaign-For-Leader (CFL: tham gia bầu thủ lĩnh) chứa tem thời gian lôgic, tới tất cả các nút dọc theo cây bao trùm. Khi TĐ CFL tới một nút lá, nút lá đó trả lời bằng cách gửi lại TĐ bầu (Voting**,** V) tới nút cha của nó. TĐ V đơn thuần xác nhận ngắn cho CFL. Nhu cầu TĐ xác nhận là điểm khác nhau căn bản giữa kiến trúc vòng logic và kiến trúc cây. Nút cha tiếp tục gửi TĐ bầu tới cha của nó sau khi đã nhận được TĐ bầu từ mọi nút con. Một nút con không bao giờ bầu trực tiếp tới CFL mà phải thông qua nút cha của nó. Một nút kết thúc trả lời của nó là hoàn thiện việc tham gia bầu thủ lĩnh. Sau đó các nút chờ công bố của thủ lĩnh mới và không tiếp nhận CFL tiếp theo khác. Nguyên nhân có thể một vài nút khởi tạo trong giai đoạn này, hai hoặc nhiều hơn TĐ CFL cùng tới một nút. Trong trường hợp này, nút nhận sẽ chọn như cha của nó nút gửi với CFL có tem thời gian bé nhất. Ràng buộc được phá vỡ theo định danh của các nút khởi tạo. Lưu ý, cha của nút có thể được thay đổi một số lần trước khi nó ủy thác cuối cùng bằng việc gửi TĐ bầu cử V mang phiếu bầu của mọi cây con của nó. Thuật toán này cho phép nút khởi tạo bầu cử đầu tiên (*có tem thời gian nhỏ nhất)* trở thành thủ lĩnh. Một nút thành công khi nhận được tất cả phiếu bầu từ các nút con của nó.

Trong mạng với lỗi hoặc thay đổi cấu hình thường xuyên, giả thiết kiến trúc luôn tồn tại cho bầu thủ lĩnh là điều không khả thi. Tuy nhiên, cây bao trùm có thể được thiết kế cho bất cứ mạng bất quy tắc nào bằng *loang TĐ*. Loang TĐ là cơ chế quảng bá trong đó mỗi nút lặp lại TĐ nhận được tới mọi kề cận của nó, ngoại trừ nút đã gửi TĐ. Cuối cùng, mọi nút trong mạng nhận được TĐ và và cây bao trùm sẽ được hình thành. Dùng cơ chế này, các nút khởi tạo loang khắp hệ thống TĐ CFL. Theo sự loang TĐ, rừng bao trùm với mỗi cây có gốc ở một nút khởi tạo được hình thành. Các TĐ tra lời V được gửi quay lui từ nút lá tới gốc. Trong tiến trình này, mỗi nút tự thay đổi cha của nó tới nút gửi CFL có tem thời gian nhỏ nhất. Nút thắng lợi khi loang là TĐ CFL với tem thời gian nhỏ nhất và nó được phép truyền bá. TĐ loang thắng lợi tiếp quản không gian của nút thua và ngay lúc đó đi tìm các nút con chưa trả lời. Khi nút khởi tạo nhận được một tem thời gian nhỏ hơn, nút khởi tạo này bị chinh phục và trở thành một nút bình thường với một nút cha. Cuối cùng, chỉ cây bao trùm với tem thời gian nỏ nhất thắng thế. Do thuật toán dùng loang và coi như không cần một kiến trúc cây cố định trước khi khởi tạo việc bầu thủ lĩnh, thuật toán loang mạnh mẽ trong các mạng nhiều lỗi.

CÂU HỎI VÀ BÀI TẬP

4.1. Các phương pháp định danh thực thể truyền thông trong dịch vụ nguyên thuỷ gửi và nhận.

4.2. Các phương án đồng bộ trong truyền thông CTĐ.

4.3. Sự chuyển đổi từ khái niệm ống dẫn sang khái niệm ống dẫn có tên mang ý nghĩa gì ?

4.4. Hãy mô tả truyền thông socket client/server hướng kết nối.

4.5. Trình bày giao thức bắt tay Handshake trong truyền thông Socket an toàn.

4.6. Truyền thông nhóm theo thứ tự tổng hai pha và so sánh với thứ tự FIFO và thứ tự nhân quả.

4.7. Khái niệm nền trong truyền thông RPC. Thi hành RPC.

4.8. Phương pháp truyền tham số và biến đổi dữ liệu trong thực hiện lời gọi RPC.

4.9. Xác thực nhau của khách và phục vụ trong truyền thông RPC.

4.10. Nội dung RPC an toàn của SUN.

4.11. Các tính chất ACID và giao thức cam kết hai pha trong truyền thông giao dịch.

4.12. Phân biệt giải pháp tên và địa chỉ. Việc chia tách thành hai giải pháp như thế có lợi điểm gì ?

4.13. Thi hành giải pháp tên.

4.14. Loại trừ ràng buộc theo cạnh tranh (sơ đồ tem - thuật toán Lamport và sơ đồ phiếu bầu)

4.15. Loại trừ ràng buộc theo thẻ bài (cấu trúc vòng, cấu trúc cây và cấu trúc quảng bá - thuật toán Suzuli/Kasami).

4.16. Bài toán bầu thủ lĩnh và thuật toán theo kiến trúc vòng lôgic.

CHƯƠNG V. LẬP LỊCH QUÁ TRÌNH PHÂN TÁN

Phương tiện TT và đồng bộ là các thành phần hệ thống thiết yếu hỗ trợ việc thực hiện đồng thời các QT tương tác. Trước khi thực hiện, QT cần phải được lên lịch (lập lịch) và định vị tài nguyên. Mục đích chính của lập lịch là *nâng cao độ đo hiệu năng tổng thể hệ thống*, chẳng hạn thời gian hoàn thành QT và tận dụng bộ xử lý. Việc tồn tại các nút xử lý phức trong hệ phân tán làm nảy sinh vấn đề thách thức cho lập lịch QT trên các bộ xử lý và ngược lại. Lập lịch không chỉ được thực hiện cục bộ trên mỗi nút mà trên toàn bộ hệ thống. Các QT phân tán có thể được thực hiện trên các nút xử lý từ xa và có thể di trú từ nút này tới nút khác để phân bố tải nhằm tăng hiệu năng. Mục đích thứ hai của lập lịch là thẹc hiện trong suốt định vị và hiệu năng bằng lập lịch QT phân tán.

Vấn đề lập lịch QT (hay công việc) đã được khảo sát rộng rãi đối với nghiên cứu điều hành. Đã có *nhiều kết quả lý thuyết* về độ phức tạp của lập lịch bộ đa xử lý. Tuy nhiên, lập lịch QT trong hệ phân tán cần đề cập cácλ chú ý thực tế thường bị bỏ qua trong phân tích lập lịch đa xử lý truyền thống. Trong hệ phân tán, tổng phí TT là đáng kể, tác dụng của hạ tầng cơ sở không thể bỏ qua và tính "động" của hệ thống phải được định vị. Các thực tế này góp phần tạo thêm sự phức tạp của lập lịch QT phân tán.

Chương này đưa ra mô hình nhằm đạt được hiệu quả hạ tầng TT và hệ thống khi lập lịch. Lập lịch QT phân tán được tổ chức thành hai nội dung: lập lịch QT tĩnh, và chia sẻ và cân bằng tải động. Thi hành thuật toán lập lịch phân tán đòi hỏi thực hiện từ xa và/hoặc năng lực di trú QT trong hệ thống. Một số vấn đề thi hành thực hiện từ xa và di trú QT được đề cập. Kết thúc chương giới thiệu hệ thống thời gian thực phân tán, trong đó lập lịch là khoảng tới hạn thời gian và xứng đáng được quan tâm đặc biệt.

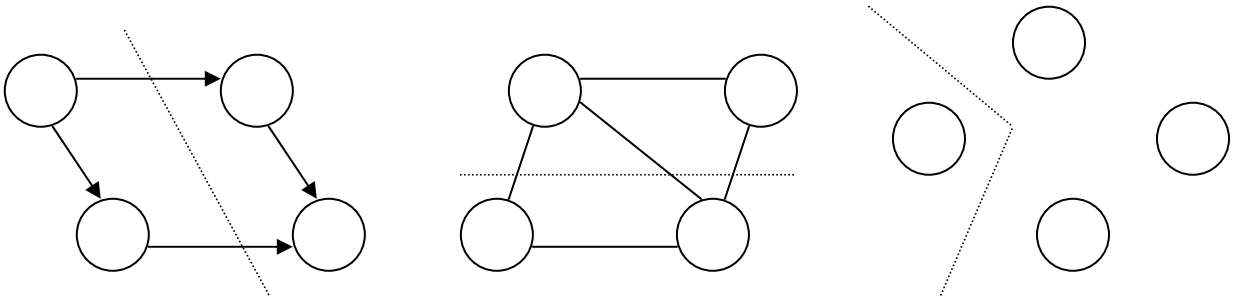
5.1. Mô hình hiệu năng hệ thống

Các thuật toán song song và phân tán được mô tả như tập QT phức được chi phối bằng các quy tắc điều chỉnh tương tác giữa các QT. Ánh xạ thuật toán vào một kiến trúc được xem xét như bộ phận của thiết kế thuật toán hoặc được xem xét một cách riêng biệt như bài toán lập lịch đối với một thuật toán cho trước và một kiến trúc hệ thống cho trước. Chương 3 sử dụng mô hình đồ thị để mô tả TT QT và tại đây xem xét tương tác QT theo mô tả tổng quát nhất theo thuật ngữ ánh xạ. Hình 5.1 cho ví dụ đơn giản về một chương trình tính toán gồm có 4 QT được ánh xạ tới một hệ thống máy tính kép với 2 bộ xử lý. Tương tác QT được biểu diễn khác nhau theo ba mô hình.

Trong mô hình QT đi trước ở hình 5.1 (a), tập QT được biểu diễn bằng một đồ thị định phướng phi chu trình (DAG-Directed Acycle Graph). Cung có hướng biểu thị quan hệ đi trước giữa các QT và chịu tổng phí truyền thông nếu các QT kết nối với nhau bằng một cung được ánh xạ tới 2 bộ xử lý khác nhau. Mô hình này được ứng dụng tốt nhất cho các QT đồng thời được sinh ra do các cấu trúc ngôn ngữ đồng thời như *cobegin*/*coend* hay *fork*/*join*. Một độ đo hữu dụng cho lập lịch tập QT như vậy là làm giảm thời gian hoàn thành bài toán xuống mức tối thiểu, bao gồm cả thời gian tính toán và TT.

Mô hình QT TT trong hình 5.1 (b) mô tả một kịch bản khác, trong đó QT được tạo ra để cùng tồn tại và truyền thông dị bộ. Cung vô hướng trong mô hình QT TT chỉ mô tả nhu cầu truyền thông giữa các QT. Do thời gian thực hiện trong mô hình là không rõ ràng nên mục tiêu lập lịch là tối ưu tổng giá truyền thông và tính toán. Bài toán được chia theo phương pháp như vậy làm giảm đến mức tối thiểu chi phí truyền thông liên-

bộ xử lý và giá tính toán của QT trên các bộ xử lý. Mô hình của QT đi trước và truyền thông là các mô hình QT tương tác.



b) Mô hình QT truyền thông c) Mô hình QT không kết nối

Hình 5.1. Phân loại quá trình

Mô hình QT độc lập ở hình 5.1(c), tương tác QT là ngầm định, và giả sử rằng các QT có thể chạy một cách độc lập và được hoàn thành trong thời gian hữu hạn. Các QT được ánh xạ tới các bộ xử lý sao cho tận dụng được các bộ xử lý một cách tối đa và làm giảm thời gian quay vòng các QT xuống đến mức nhỏ nhất. Thời gian quay vòng các QT được xác định như tổng thời gian thực hiện và xếp hàng do phải chờ các QT khác. Trong trường hợp động, cho phép QT "di trú" giữa các bộ xử lý để đạt hiệu quả trong chia xẻ và cân bằng tải. Nếu QT được phép di trú từ nút có tải lớn đến nút có tải nhỏ thì định vị ban đầu các QT là chưa tới hạn. Hơn nữa, hiệu năng được cải tiến đáng kể do lịch các QT trở nên thích ứng với sự thay đổi tải hệ thống. Chia xẻ và cân bằng tải không hạn chế các QT độc lập. Nếu QT truyền thông với một QT khác thì chiến lược "di trú" nên chú ý cân bằng các thay đổi trong các nhu cầu truyền thông giữa các bộ xử lý do thay đổi bộ xử lý và lợi ích từ chia xẻ tải.

Phân hoạch bài toán thành nhiều QT để giải làm thời gian hoàn thành bài toán nhanh hơn. *Tăng tốc* được coi như độ đo hiệu năng là mục tiêu đáng quan tâm trong thiết kế các thuật toán song song và phân tán. Tăng tốc tính toán là một hàm của thiết kế thuật toán và hiệu quả của thuật toán lập lịch ánh xạ thuật toán vào kiến trúc hệ thống hạ tầng. Dưới đây đưa ra một mô hình tăng tốc mô tả và phân tích mối quan hệ giữa *thuật toán*, *kiến trúc hệ thống* và *lịch thực hiện*. Trong mô hình này, hệ số tăng tốc S là hàm của thuật toán song song, kiến trúc của hệ thống và lịch thực hiện, được biểu diễn theo công thức:

S = F(thuật toán, hệ thống, lịch).

S có thể được viết như sau:

$$S = \frac{OSPT}{CPT} = \frac{OSPT}{OCPT_{ideal}} x \frac{OCPT_{iedal}}{CPT} = S_i x S_d$$

Trong đó :

+ OSPT (Optimal Sequential Processing Time): thời gian tốt nhất có thể đạt được trên bộ đơn xử lý sử dụng thuật toán tuần tự tốt nhất.

+ CPT (Concurrent Processing Time): thời gian thực sự đạt được trên một hệ n-bộ xử lý cùng với thuật toán đồng thời và một phương pháp lập lịch cụ thể đang được xem xét.

+ $OCPT_{ideal}$(Optimal Concurrent Processing Time on an ideal system): thời gian tốt nhất có thể đạt được với cũng thuật toán đồng thời được xem xét trên một hệ n-bộ

xử lý lý tưởng (một hệ thống không tính tới tổng phí truyền tin giữa các bộ xử lý) và đã được lên chương trình bằng một phương thức lập lịch tối ưu nhất.

+ S_i: độ tăng tốc lý tưởng đạt được nhờ sử dụng hệ đa xử lý phức so với thời gian tuần tự tốt nhất.

+S_d: độ hao phí của hệ thống thực hiện trên thực tế so với hệ thống lý tưởng.

Để nhận rõ vai trò của thuật toán, hệ thống và lập lịch, công thức biểu diễn độ tăng tốc được rút gọn hơn. S_i có thể được viết lại như sau:

$$S_i = \frac{RC}{RP} * n \qquad \text{trong đó: } RP = \frac{\sum_{i=1}^{m} P_i}{\text{OSPT}} \qquad \text{và } RC = \frac{\sum_{i=1}^{m} P_i}{OCPT_{ideal} * n}$$

và n là số lượng bộ xử lý. Đại lượng $\sum_{i=1}^{m} P_i$ là tổng số các thao tác tính toán của thuật toán đồng thời, trong đó *m* là số bài toán con trong thuật toán. Đại lượng này thường lớn hơn *OSPT* do các mã phụ có thể được bổ sung khi biến đổi thuật toán tuần tự thành thuật toán đồng thời. S_d có thể được viết lại như sau:

$$S_d = \frac{1}{1+\rho} \qquad \text{trong đó, } \rho = \frac{CPT - OCPT_{iedal}}{OCPT_{iedal}}$$

Trong biểu thức S_i, RP là yêu cầu xử lý liên quan (Relative Processing), là tỷ số giữa tổng số thời gian tính toán cần thiết cho thuật toán song song so với thời gian xử lý của thuật toán tuần tự tối ưu. Nó cho thấy lượng hao phí tăng tốc do thay thế thuật toán tuần tự tối ưu bằng một thuật toán thích hợp thực hiện đồng thời nhưng có thể có tổng nhu cầu xử lý lớn hơn. *RP* khác với S_d ở chỗ *RP* là lượng thời gian hao phí của thuật toán song song do việc thay đổi thuật toán, trong khi S_d là lượng thời gian hao phí của thuật toán song song do việc thi hành thuật toán.

Độ đo đồng thời liên quan *RC* (Relative Concurency) đo mức độ sử dụng tốt nhất của hệ n-bộ xử lý. Nó cho thấy bài toán đã cho và thuật toán dành cho bài toán tốt như thế nào đối với hệ n-bộ xử lý lý tưởng. *RC*=1 tương ứng với việc sử dụng các bộ xử lý là tốt nhất. Một thuật toán đồng thời tốt là thuật toán làm cho *RP* đạt giá trị nhỏ nhất và *RC* đạt giá trị lớn nhất. Biểu thức cuối cùng cho tăng tốc *S* là:

$$S = \frac{RC}{RP} \times \frac{1}{1+\rho} \times n$$

Tóm lại, nhân tố tăng tốc S là một hàm của RC (tổn thất lý thuyết khi song song hóa), *RP* (lượng bổ sung vào tổng nhu cầu tính toán), ρ (thiếu hụt song song hóa khi thi hành trên một máy thực) và *n* (số bộ xử lý được sử dụng).

Số hạng ρ được goi là hiệu suất tổn thất, được xác định như tỷ số giữa tổng phí theo hệ thống thực nói trên theo mọi nguyên nhân đối với thời gian xử lý tối ưu lý tưởng. Nó là hàm của lập lịch và kiến trúc hệ thống. Rất hữu dụng khi phân tích ρ thành 2 số hạng riêng biệt : $\rho = \rho_{syst} + \rho_{sched}$, tương ứng với hiệu suất hao phí do hệ thống và lịch gây ra. Tuy nhiên, điều này không dễ thực hiện do lịch và hệ thống phụ thuộc vào nhau. Do tổng phí TT đôi lúc bị che khuất và chồng chéo lên các QT tính toán khác trong lập lịch nên có thể không ảnh hưởng tới tổn thất hiệu suất. Đây là một điểm chính trong lập lịch QT có tính đến tổng phí TT giữa các bộ xử lý. Một lịch tốt là lịch hợp lý nhất trên hệ thống đã cho sao cho nó có khả năng che dấu được tổng phí càng nhiều càng

tốt. Đoạn tiếp theo minh hoạ về sự phụ thuộc lẫn nhau giữa hai yếu tố lập lịch và hệ thống và phân tích sơ bộ hai yếu tố này.

Giả sử X mô tả một hệ đa máy tính đang được nghiên cứu và *Y′* mô tả một chiến lược lập lịch được mở rộng cho hệ thống *X* từ chiến lược lập lịch *Y* trên hệ thống lý tưởng tương ứng. *CPT(X,Y′)* và $CPT_{iedal}\ (Y)$ tương ứng là các thời gian quá trình đồng thời cho máy *X* theo các chiến lược lập lịch *Y′* và *Y* tương ứng. Có thể biểu diễn hiệu suất hao phí ρ như sau:

$$\rho = \frac{CPT(X,Y^{'}) - OCPT_{iedal}}{OCPT_{iedal}}$$

$$= \frac{CPT(X,Y') - CPT_{iedal}(Y)}{OCPT_{iedal}} + \frac{CPT_{iedal}(Y) - OCPT_{iedal}}{OCPT_{iedal}}$$

$$= \rho_{syst} + \rho^{'}_{sched}$$

Tương tự, chúng ta làm ngược quá trình phân tích theo giải tích tổn thất hiệu quả lập lịch không tối ưu trước khi giải tích hiệu quả của hệ thống không lý tưởng. Như thế,

$$\rho = \frac{CPT(X,Z) - OCPT_{iedal}}{OCPT_{iedal}}$$

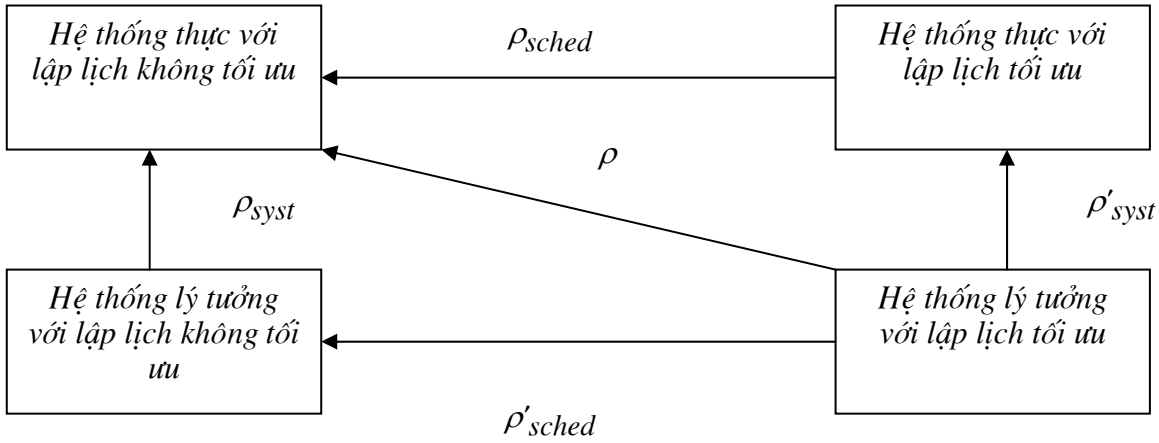
$$= \frac{CPT(X,Z) - OCPT(X)}{OCPT_{iedal}} + \frac{OCPT(X) - OCPT_{iedal}}{OCPT_{iedal}}$$

$$= \rho_{sched} + \rho^{'}_{syst}$$

Hình 5.2 phân tích tường minh hiệu suất hao phí do lập lịch và TT trong hệ thống gây ra. Ảnh hưởng đáng kể của TT trong hệ thống được định vị cẩn thận khi thiết kế các thuật toán lập lịch phân tán.

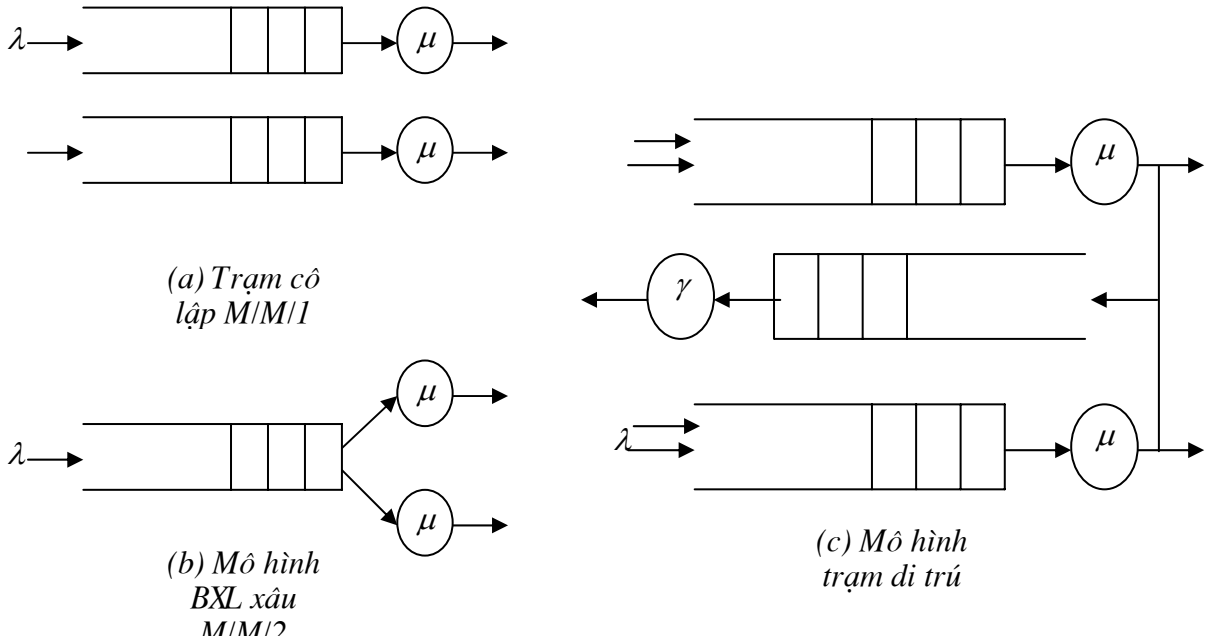
Mô hình tăng tốc chung tích hợp 3 thành phần chính: phát triển thuật toán, kiến trúc hệ thống và chiến lược lập lịch, với mục đích làm giảm đến mức tối thiểu tổng thời gian hoàn thành (makespan) của tập các QT tương tác. Nếu các QT không bị ràng buộc bởi quan hệ đi trước và được tự do phân phối lại hoặc được di trú dọc theo các bộ xử lý trong hệ thống thì hiệu năng được cải tiến hơn nữa nhờ chia xẻ tải. Điều đó có nghĩa là, các QT có thể được di trú từ những nút có tải lớn tới những nút rỗi (nếu tồn tại các nút đó). Có thể được tiến thêm một bước xa hơn khi tới chia xẻ tải giữa tất cả các nút sao cho ***càng đều càng tốt***, bằng phương pháp tĩnh hoặc động. Phân bố tải tĩnh được gọi ***chia xẻ tải***, và phân bố tải động được gọi là ***cân bằng tải***. Lợi ích của phân bố tải là các bộ xử lý được tận dụng triệt để hơn và cải tiến được thời gian quay vòng các QT. Di trú QT rút gọn thời gian xếp hàng, kể cả giá tăng thêm theo tổng phí TT.

Mục đích của chia xẻ tải trong hệ phân tán là làm hoàn toàn rành mạch. Điều đó cũng phù hợp với bất kỳ việc khởi tạo máy tính gồm nhiều nút xử lý được ghép nối lỏng, luôn có một số nút có tải lớn và một số nút có tải nhỏ, nhưng phần lớn các nút là hoàn toàn không tải. Để tận dụng hơn về năng suất xử lý, các QT có thể được gửi tới các bộ xử lý rỗi theo phương pháp tĩnh ngay khi chúng vừa xuất hiện (tương ứng với mô hình bộ xử lý xâu) hoặc "di trú" theo phương pháp động từ những bộ xử lý có tải lớn đến những bộ xử lý có tải nhỏ (tương ứng với mô hình trạm làm việc). Thời gian quay vòng QT cũng được cải tiến.



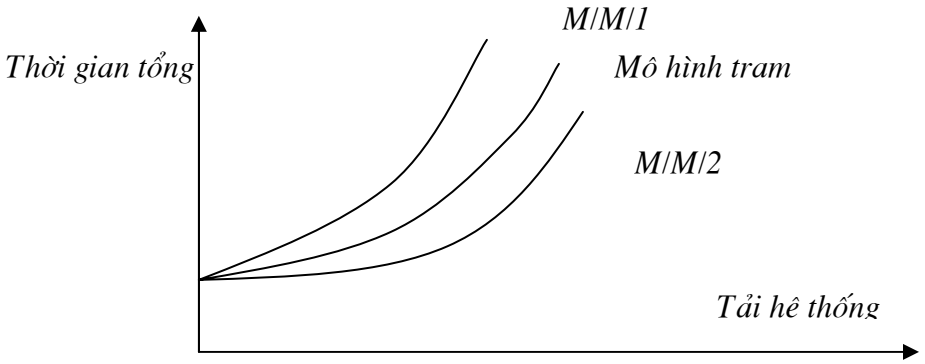
Hình 5.2. Tổn thất hiệu quả theo lập lịch và TT

Hình 5.3 trình bày hai mô hình hàng đợi đơn giản về môi trường phân tán theo bộ xử lý xâu và theo trạm làm việc so sánh với hệ thống các trạm làm việc cô lập với đường tham chiếu (baseline). Để rõ ràng, trong các mô hình này chỉ gồm hai nút xử lý. Trong mô hình bộ xử lý xâu, một QT được gửi tới một bộ xử lý phù hợp và ở lại đó trong suốt thời gian thực hiện nó.



Hình 5.3. Mô hình hàng đợi BXL xâu và trạm làm việc

Hiệu năng hệ thống được mô tả theo mô hình dòng xếp hàng có thể tính được nhờ sử dụng kiến thức toán học như lý thuyết hàng đợi. Sử dụng kí hiệu chuẩn Kendall để mô tả tính chất thống kê của hàng đợi. Hàng đợi *X/Y/c* là một QT X xuất hiện, một phân bố thời gian phục vụ *Y,* và *c* máy phục vụ. Ví dụ, có thể mô tả bộ xử lý xâu như hàng đợi M/M/2. M tuân theo phân bố Markov, là loại phân bố dễ xử lý khi phân tích. Mô hình hệ thống với hai máy phục vụ trong đó công việc đợi xử lý có thể được phục vụ trên một bộ xử lý bất kỳ. Tổng quát, mô hình hóa bộ xử lý xâu là hàng đợi M/M/k.



Hình 5.4. So sánh hiệu năng theo chia xẻ tải

Trong mô hình trạm làm việc di trú, các QT được phép dịch chuyển từ trạm này tới trạm khác. Quyết định di trú QT vào lúc nào, ở đâu, như thế nào sẽ được xem xét sau và chưa được trình bày tường minh trong hình vẽ. Di trú QT phải chịu độ trễ truyền thông được lấy mẫu bởi một hàng đợi truyền thông do một kênh truyền thông phục vụ. Tỷ số di trú γ là hàm của dải thông kênh truyền, giao thức di trú QT, và quan trọng hơn là ngữ cảnh và thông tin trạng thái của QT đang được chuyển giao.

Hình 5.4 chỉ ra lợi ích của phân bố (hoặc phân bố lại) tải trong các mô hình bộ xử lý xâu và trạm làm việc. Các cận trên và cận dưới cho thời gian quay vòng quá trình trung bình được trình bày bằng hai phương trình của mô hình M/M/1 và M/M/2:

$$TT_1 = \frac{1}{\mu - \lambda}$$

$$TT_2 = \frac{\mu}{(\mu + \gamma)(\mu + \lambda)}$$

TT_1 là thời gian quay vòng trung bình, với λ và μ là tần suất xuất hiện QT và tần suất được phục vụ của mỗi nút xử lý. Công thức liên quan có thể tìm thấy trong lý thuyết hàng đợi cổ điển. Hiệu năng trong mô hình trạm làm việc với tổng chi phí TT nằm giữa M/M/1 (không có chia xẻ tải) và M/M/2 (mô hình bộ xử lý xâu lý tưởng với tổng phí TT là không đáng kể). Tỷ lệ di trú QT γ thay đổi từ 0 đến ∞, tương ứng với hiệu năng tiệm cận của M/M/1 và M/M/2.

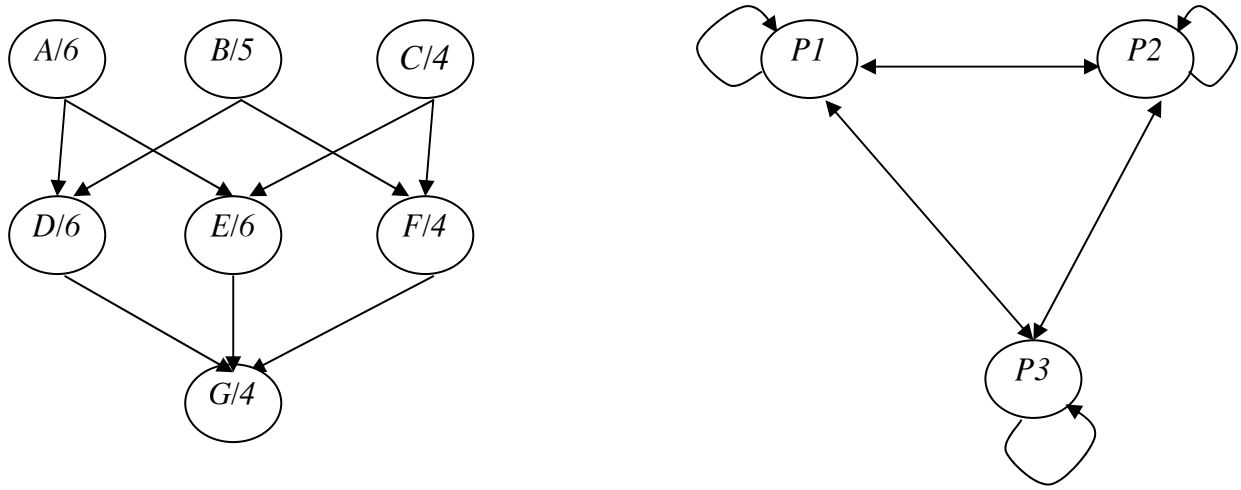
5.2. Lập lịch quá trình tĩnh

Lập lịch QT tĩnh (lý thuyết lập lịch tiền định) đã được nghiên cứu rộng rãi. Bài toán đặt ra là lập lịch cho một tập thứ tự bộ phận các bài toán trên hệ thống đa xử lý với các bộ xử lý giống nhau nhằm mục tiêu giảm thiểu toàn bộ thời gian hoàn thiện (makespan). Có nhiều công trình tổng quan xuất sắc, trong đó có bài viết của Coffman và Graham. Các nghiên cứu trong lĩnh vực này chỉ ra rằng, tuy có các trường hợp giới hạn (chẳng hạn, lập lịch các bài toán có thời gian thực hiện đơn vị hay mô hình song xử lý), bài toán lập lịch tối ưu là độ phức tạp NP-đầy đủ. Bởi vậy, hầu hết các nghiên cứu định hướng sử dụng phương pháp xấp xỉ hay phương pháp heuristic nhằm đi tới giải pháp gần tối ưu cho vấn đề này. Hệ thống tính toán hạ tầng của bài toán cổ điển với các giả thiết không có chi phí liên QT đưa đến cạnh tranh truyền thông và bộ nhớ. Giả thiết này có thể hợp lý với kiến trúc đa xử lý nào đó. Tuy nhiên, nó không có giá trị đối với hệ thống phân tán CTĐ hoặc mạng máy tính, trong đó TTLQT không những không thể bỏ qua mà còn là một đặc trưng quan trọng của hệ thống. Do quá thô bạo khi bỏ qua chú ý TT, với những hệ thống chi phí TT là không thể bỏ qua được, tập trung vào các tiệm cận heristic tốt nhưng dễ dàng thi hành để lập lịch QT trong hệ phân tán.

Một thuật toán lập lịch phân tán heuristic tốt là nó phải cân bằng tốt và giảm thiểu sự chồng chéo trong tính toán và truyền thông. Khảo sát hai bài toán lập lịch đặc biệt, một là lập lịch tất cả QT trong một bộ xử lý đơn và hai là mỗi bộ xử lý được phân công tới mỗi QT. Ở bài toán đầu tiên, tuy không có chi phí truyền thông liên kết nên cũng không cần có tính đồng thời. Bài toán thứ hai tuy thể hiện tốt tính đồng thời nhưng vướng mắc phí tổn truyền thông. Đối tượng lập lịch của chúng ta cần thống nhất giữa việc hạn chế tối đa tắc nghẽn và chi phí truyền thông, đạt sự đồng thời cao nhất có thể tại cùng một thời điểm.

Trong lập lịch tĩnh, ánh xạ các QT tới các bộ xử lý phải được xác định trước khi thực hiện các QT đó. Ngay khi QT bắt đầu, nó được lưu lại trong bộ xử lý cho đến khi hoàn tất. Không bao giờ có ý định di chuyển nó tới bộ xử lý khác để thực hiện. Một thuật toán lập lịch tốt đòi hỏi hiểu biết tốt về hành vi của QT, chẳng hạn như thời gian thực hiện QT, mối quan hệ đi trước và thành phần truyền thông giữa các QT. Những thông tin này có thể là tìm thấy trong bộ biên dịch của ngôn ngữ đồng thời. Quyết định lập lịch là tập trung và không thích nghi. Đây cũng là một số mặt hạn chế của lập lịch tĩnh.

Trong hai phần sau đây, chúng ta xem xét ảnh hưởng của truyền thông trong lập lịch tĩnh, sử dụng mô hình đi trước và mô hình QT truyền thông.



(a) Mô hình QT đi trước

(b) Mô hình HT truyền thông

Hình 5.5. Mô hình hệ thống truyền thông và quá trình đi trước

5.2.1. Mô hình quá trình đi trước

Mô hình QT đi trước trong hình 5.1 (a) được sử dụng trong lập lịch đa xử lý tĩnh mà mục tiêu căn bản là tối thiểu hoá toàn bộ thời gian hoàn thành. Trong mô hình QT đi trước, một chương trình được trình bày bằng một DAG. Mỗi một nút trong hình vẽ biểu thị một nhiệm vụ được thực hiện trong một khoảng thời gian xác định. Mỗi cung nối biểu thị quan hệ đi trước giữa hai nhiệm vụ và được gán nhãn là trọng số biểu diễn số đơn vị TĐ được chuyển tới công việc tiếp sau khi hoàn thành công việc. Hình 5.5 a là ví dụ của chương trình DAG, bao gồm 7 nhiệm vụ (từ A đến G) cùng với việc chỉ rõ thời gian thực thi các nhiệm vụ đó là số đơn vị TĐ truyền thông giữa những nhiệm vụ với nhau. Kiến trúc hạ tầng trên đó các nhiệm vụ nền được thiết lập được đặc trưng bằng mô hình hệ thống truyền thông chỉ rõ giá thành truyền thông đơn vị giữa các bộ xử lý. Hình 5.5 b là một ví dụ của một mô hình hệ thống truyền thông cùng với ba bộ xử lý (P1, P2, P3). Giá thành truyền thông đơn vị thường là đáng kể với truyền thông đa xử lý và không đáng kể (không trọng lượng trong các đường nối nội tại) đối với

truyền thông nội bộ. Mô hình này rất đơn giản, nó giữ truyền thông mà không cần đưa ra chi tiết cấu trúc phần cứng. Giá thành truyền thông giữa hai nhiệm vụ được tính bằng tích đơn vị giá thành truyền thông trong đồ thị hệ thống truyền thông với số đơn vị TĐ trong đồ thị xử lý ưu tiên. Ví dụ, nhiệm vụ A và E trong hình 5.5 được lập lịch tương ứng trên bộ xử lý P1 và P3, giá thành truyền thông là 8 = 2*4. Raywayd – Smith đưa ra khảo sát mô hình tương tự nhưng với một số hạn chế trong tất cả các QT có đơn vị tính toán và thời gian truyền thông. Thậm chí với một giả thiết đơn giản thì việc tìm giá trị tối thiểu của toàn bộ thời gian hoàn thành là NP-complete. Vì vậy chúng ta sẽ ứng dụng thuật toán heuristic cho việc tìm kiếm một ánh xạ tốt từ mô hình QT tới mô hình hệ thống.

Nếu bỏ qua phí tổn đường truyền, chúng ta xem xét phương pháp "heuristic tham ăn" đơn giản: chiến lược LS (lập lịch danh sách). Không một bộ xử lý nào đặt ở chế độ nhàn rỗi nếu còn những tác vụ có thể cần xử lý. Đối với DAG trong hình 5.5 a, kết quả lập lịch trong hình 5.6 a. Tổng thời gian hoàn thành là 16 đơn vị. Đối với đồ thị QT đi trước, khái niệm về đường tới hạn là rất có ích. Đường tới hạn là đường thực hiện dài nhất trong DAG, nó lại là đường ngắn nhất của toàn bộ thời gian hoàn tất. Đường tới hạn rất quan trọng trong nội dung lập lịch. Nó được sử dụng thường xuyên để phân tích việc thực thi một thuật toán "heuristic". Đường tới hạn trong đồ thị hình 5.5 a là (ADG và AEG) độ dài 16 = 6+6+4. Vì vậy, LS trong hình 5.6 a (tổng thời gian hoàn thành cũng là 16) là tốt ưu nhất ngay khi tìm ra thuật toán. Một số thuật toán lập lịch được tìm ra cũng dựa vào đường tới hạn bắt nguồn từ tính ưu tiên cho những nhiệm vụ. Một số chiến lược lập lịch được tìm ra đơn giản là vạch ra tất cả công việc trong đường tới hạn lên một bộ xử lý đơn. Ví dụ trong hình 5.5 a, những nhiệm vụ A,D và G trên đường tới hạn được vạch tới bộ xử lý P1.

(a) LS

P1	A/6		D/6	G/4
P2	B/5		F/4	7
P3	C/4	2	E/6	4

Tổng chi phí là 16

(b) ELS

P1	A/6	2	D/6	10	G/4
P2	B/5	2	F/4	17	
P3	C/4	10		E/6	8

Tổng chi phí là 28

(c) ETF

P1	A/6		E/6	6
P2	B/5	2	D/6	G/4
P3	C/4	4	F/4	6

Tổng chi phí là 18

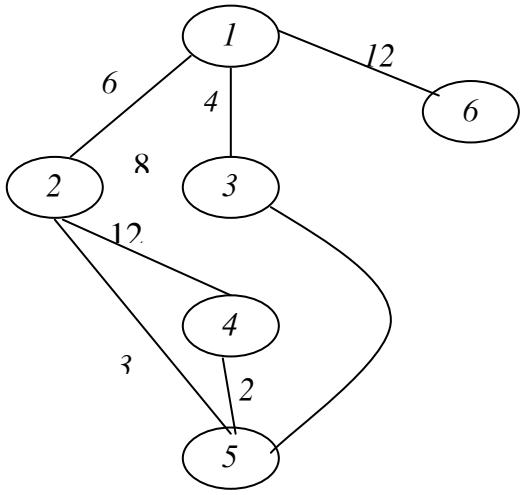
Hình 5.6.

Nếu tính đến phí tổn đường truyền, chúng ta có thể mở rộng việc lập lịch các danh sách trực tiếp (LS). Lập lịch các danh sách mở rộng đầu (ELS) đầu tiên thực hiện chỉ định những công việc tới bộ xử lý bằng việc cung tấp LS như khi hệ thống rỗi trong truyền thông liên kết. Nó thêm vào thời gian trễ truyền thông khi cần thiết để lập lịch

được chứa bởi LS. Những trì hoãn truyền thông được tính toán bởi việc nhân giá thành đơn vị truyền thông và những đơn vị thông báo. Kết quả ELS cho cùng một vấn đề lập lịch có tổng thời gian hoàn thành là 28 đơn vị, như trình bày trong hình 5.6 b Dashed – lines trong hình biểu diễn QT đợi truyền thông (giá thành đơn vị truyền thông được nhân bởi số lượng các đơn vị thông báo).

Chiến lược ELS không thể đạt tối ưu. Vấn đề cơ bản là việc quyết định lập lịch đã được thiết lập mà không được báo trước trong việc truyền thông. Thuật toán có thể được cải tiến khi chúng ta trì hoãn quyết định lâu nhất cho đến khi chúng ta biết nhiều hơn về hệ thống. Theo chiến lược tham ăn này chúng ta có phương pháp lập lịch ưu tiên tác vụ đầu tiên (ETF), tác vụ sớm nhất phải được lập lịch đầu tiên. Sử dụng chiến lược này trong cùng một ví dụ, chúng ta sẽ trì hoãn lập lịch tác vụ F bởi tác vụ E sẽ trở thành lập lịch đầu tiên nếu trì hoãn truyền thông cũng liên quan đến việc tính toán. Lập lịch ETF trong hình 5.6 c đưa ra kết quả tốt hơn là tổng thời gian hoàn thành là 18 đơn vị.

Mô hình QT và hệ thống là khá rõ ràng để mô hình hoá bài toán quá trình lập lịch trong DAG vào hệ thống với sự trễ truyền thông. Ví dụ chỉ ra rằng một lịch tối ưu cho hệ thống này không nhất thiết là lịch tốt cho hệ thống khác đồng thời với cấu trúc truyền thông khác nhau. Lập lịch tốt hơn có thể đạt được nhờ trộn nhau giữa truyền thông với tính toán và vì vậy che dấu hiệu quả tổng phí truyền thông. Khái niệm đường tới hạn có thể được dùng để hỗ trợ việc che dấu truyền thông (thu hút tổng phí truyền thông vào đường tới hạn). Bất kỳ đường tính toán ngắn hơn đường tới hạn được thu vào tổng phí TT nào đó để chập với một tính toán khác mà không ảnh hưởng đến tổng thời gian hoàn thiện.



Hình 5.7. Giá tính toán và đồ thị truyền thông

5.2.2. Mô hình quá trình truyền thông

Mô hình đồ thị đi trước biểu diễn QT được thảo luận trong phần trước là mô hình tính toán. Chương trình được biểu diễn bằng DAG là những ứng dụng người dùng điển hình, trong đó ràng buộc đi trước giữa các bài toán trong chương trình được người dùng chỉ dẫn rõ ràng. Mục tiêu cơ bản của lập lịch là đạt sự đồng thời tối đa việc thực hiện bài toán trong chương trình. Giảm tối thiểu truyền thông bài toán đóng vai trò thứ yếu, mặc dù có ảnh hưởng đáng kể tới số hiệu hiệu năng chính: thời gian hoàn thiện tổng thể.

Lập lịch QT cho những ứng dụng hệ thống theo nhiều bối cảnh rất khác nhau, bởi vì các QT trong một ứng dụng hệ thống có thể tạo ra một cách độc lập. Không có ràng buộc trước-sau ngoại trừ nhu cầu truyền thông giữa các QT. Không có thời gian hoàn thành của các QT như trường hợp mô hình QT đi trước. Mục tiêu của lập lịch QT là tận

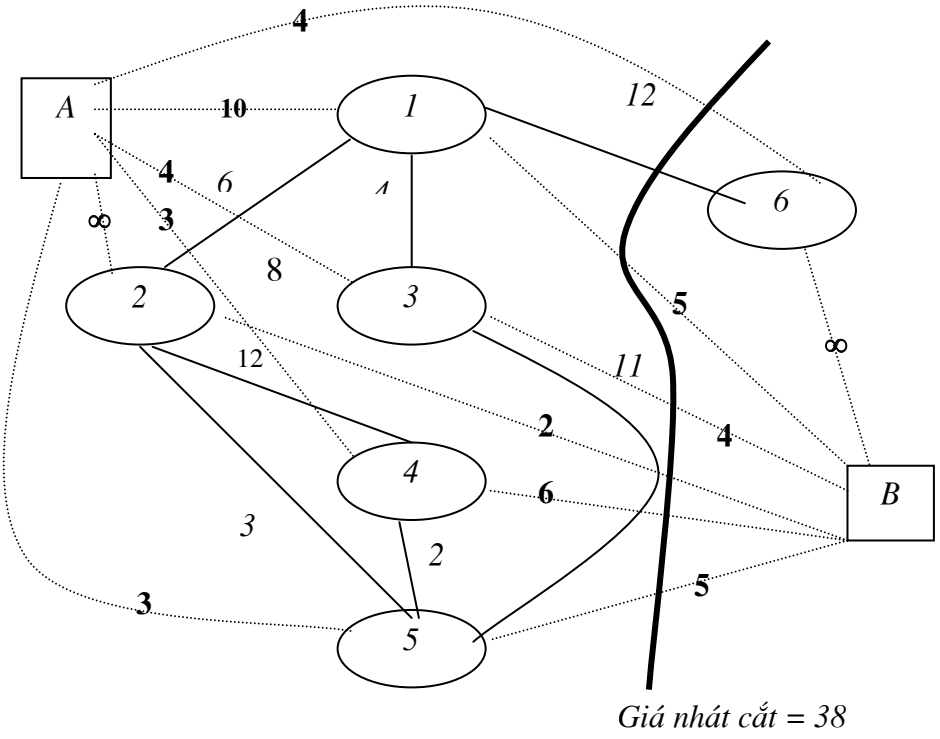
dụng tối đa nguồn tài nguyên và giảm tối thiểu truyền thông liên QT. Những ứng dụng này là mô hình tốt nhất cho mô hình QT truyền thông, được trình bày trong hình 5.1 b.

Mô hình QT truyền thông được biểu diễn bằng một đồ thị vô hướng G với tập V các đỉnh biểu diễn QT và tập E các cạnh có trọng số nối hai đỉnh biểu diễn số lượng giao dịch của hai QT liên kết nhau. Giả thiết về việc thực hiện QT và truyền thông là tương tự mô hình đi trước song có một sự khác biệt nhỏ. Việc thực hiện QT và truyền thông được biểu diễn theo giá thành.

Giá thành thực hiện QT là hàm theo BXL mà QT được gán tới đo để thực hiện. Vấn đề chính yếu là các bộ xử lý không đồng nhất (khác nhau về tốc độ và cấu trúc phần cứng). Do vậy, dùng ký hiệu e_j (p_i) để biểu thị giá thành cho QT j trên p_i, trong đó p_i là bộ xử lý được dùng cho QT j. Giá thành truyền thông $c_{i,j}$ (p_i, p_j) giữa hai QT i và j dùng cho hai bộ xử lý khác nhau p_i và p_j là tỉ lệ với trọng số cung kết nối i với j. Giá thành truyền thông được xem là không đáng kể (giá thành bằng 0) khi i =j. Bài toán được đặt ra là tìm phân công tối ưu của mô hình m mođun QT tới P bộ xử lý theo mối quan hệ của hàm đối tượng dưới đây được gọi là *Bài toán định vị mođun*.

$$Cost\ (G, P) = \sum_{J \in V(G)} e_j(p_i) + \sum_{(i,j) \in E(G)} e_{i,j}(p_i, p_j)$$

Bài toán định vị mođun được Stone đưa ra đầu tiên và được nghiên cứu rộng rãi khá lâu. Tương tự như phần lớn các ứng dụng đồ thị, Bài toán định vị môđun tổng quát là NP-đầy đủ ngoại trừ một vài trường hợp hạn chế. Với P=2, Stone dự đoán một cách giải đa thức hiệu quả sử dụng thuật toán dòng - cực đại (maximum – flow) của Ford-Fulkerson. Các thuật toán giải đa thức cũng đã được phát triển bởi Bokhari và Towsley cho một vài đồ thị tôpô đặc biệt như đồ thị dạng cây và song song chuỗi. Trong ví dụ dưới đây chúng ta minh họa khái niệm trên bằng xem xét mô hình hàng hoá song xử lý cuả Stone trong việc phân chia đồ thị QT truyền thông tới kiến trúc để đạt được tổng giá thành thực hiện và truyền thông nhỏ nhất.



Hình 5.8. Nhắt cắt giá tối thiểu

Khảo sát một chương trình bao gồm 6 QT sẽ được lập lịch vào hai bộ xử lý A và B nhằm giảm tối thiểu giá thành tổng tính toán và truyền thông. Thời gian thực hiện cho mỗi QT trên mỗi bộ xử lý được trình bày qua hình 5.7 a. Hình 5.7 b là đồ thị biểu diễn đa xử lý truyền thông giữa 6 QT. Hai bộ xử lý là không giống nhau. Ví dụ QT 1 cần 5 đơn vị giá thành để chạy trên bộ xử lý A nhưng cần 10 đơn vị giá thành khi chạy trên bộ xử lý B. Nhãn gán trên một cạnh của đồ thị truyền thông là giá thành truyền thông

nếu hai QT kết nối nhau được định vị tới những bộ xử lý khác nhau. Để ánh xạ QT tới các bộ xử lý, phân chia thành hai đồ thị rời nhau bằng một đường kẻ cắt ngang qua một số cung. Kết quả phân chia thành hai đồ thị rời nhau, mỗi đồ thị gán tới một bộ xử lý. Tập các cung bị loại bỏ qua nhát cắt được gọi là *tập cắt (cut set)*. Giá thành của một *tập cắt* là tổng trọng lượng của những cung biểu thị chính tổng giá thành truyền thông liên QT giữa hai bộ xử lý.

Bài toán tối ưu sẽ là tầm thường khi chúng ta chỉ phải giảm tối thiểu giá thành truyền thông vì chúng ta có thể sắp đặt tất cả các QT lên một bộ xử lý đơn và loại trừ tất cả trần các truyền thông liên QT. Tối ưu là vô nghĩa trừ phi cần phải đảm bảo các ràng buộc nào đó trong việc tính toán thực hiện và thi hành khác. Điều kiện hạn chế là QT nào đó chỉ có thể chạy được trên một bộ xử lý nào đó như hình 5.7 a là một ví dụ tốt về ràng buộc tính toán. Một vài việc thực thi có thể yêu cầu không nhiều hơn k QT chỉ định cho một bộ xử lý hay những QT đó được chỉ định tới tất cả các bộ xử lý hiện có.

Hình 5.8 chỉ ra nhất cắt giá thành tối thiểu cho trường hợp hình 5.7 với hàm tính giá COST (G, P). Trong lược đồ, bổ sung hai đỉnh mới biểu diễn các bộ xử lý A và B vào đồ thị truyền thông (cùng những cung nối mỗi bộ xử lý tới mỗi đỉnh QT). Trọng số được gán tới cạnh nối giữa bộ xử lý A và QT i là giá thành thực hiện QT i trên bộ xử lý B và ngược lại. Việc gán trọng số kiểu này là khôn ngoan bởi vì một vết cắt dọc theo đường đậm nét liên quan đến phân công QT được thực hiện trên bộ xử lý B. Chúng ta xem xét chỉ các nhát cắt phân chia các nút (A và B). Tổng trọng số của các đường nối trong vết cắt là tổng giá thành truyền thông và giá thành tính toán.

Việc tính tập cắt giá thành tối thiểu cho mô hình trên là tương đương với việc tìm dòng cực đại (maximum-flow) và cắt tối thiểu (minimum-cut) của mạng hàng hóa. Đồ thị ở hình 5.8 có thể hiểu như một mạng với các đường giao thông (cung) nối các thành phố (đỉnh) với nhau. Trọng số trên đường nối là thông lượng của đoạn. Nút A là thành phố nguồn và nút B là thành phố đích của việc vận chuyển hàng hoá. Khi cho một đồ thị hàng hoá, vấn đề tối ưu là tìm ra luồng cực đại từ nguồn tới đích. Fort và Fulkerson trình bày một thuật toán gán nhãn cho phép tìm một cách hệ thống đường mở rộng dần từ nguồn tới đích (thuật toán được thấy trong hầu hết các cuốn sách giáo khoa về thuật toán). Hai ông cũng chứng minh rằng luồng cực đại (maximum fow) cho một mạng tương đương mặt cắt nhỏ nhất (minimum cut) làm tách rời nguồn với đích trong đồ thị. Thuật toán luồng cực đại và định lý mặt cắt nhỏ nhất của luồng cực đại hoàn toàn phù hợp với sự tối ưu hoá bài toán định vị mô - đun (sự lập lịch QT) cho hai bộ xử lý.

Để tổng quát hoá bài toán có nhiều hơn hai bộ xử lý, Stone phác thảo giải pháp cho hệ thống có 3 bộ xử lý và đề xuất một phương pháp lặp sử dụng thuật toán cho hai bộ xử lý để giải quyết những bài toán có n bộ xử lý. Để tìm ra một sự định vị mô-đun của m QT cho n bộ xử lý, thuật toán mặt cắt nhỏ nhất luồng cực đại có thể áp dụng cho một bộ xử lý P_i và một bộ siêu xử lý ảo P bao gồm các bộ xử lý còn lại. Sau khi vài QT đã được lên lịch cho P_i, thủ tục được lặp lại tương tự trên bộ siêu xử lý cho đến khi tất cả các QT được ấn định.

Bài toán định vị mô-đun là phức tạp vì những mục đích của sự tối ưu hoá cho việc giảm chi phí tính toán và truyền tin xuống mức thấp nhất thường mâu thuẫn (đối lập) với nhau. Bài toán đủ quan trọng để chứng minh cho những giải pháp mang tính kinh nghiệm (tự tìm tòi). Một phương pháp để phân chia sự tối ưu hoá tính toán và truyền thông trở thành 2 vấn đề riêng biệt. Trong một mang máy tính nơi chi phí truyền tin có thể có ý nghĩa (đáng kể) hơn chi phí tính toán, ta có thể kếtp hợp các QT với sự tương tác giữa các QT bậc cao thành các nhóm QT. Các QT trong mỗi nhóm sau khi được ấn định cho bộ xử lý sẽ làm giảm chi phí tính toán xuống mức thấp nhất. Sự hợp nhất các QT truyền tin giữa các bộ xử lý đơn giản nhưng có thể thực hiện được nhiều phép tính

hơn trên bộ xử lý và do đó làm giảm bớt sự trùng lặp. Một giải pháp đơn giản là chỉ kết hợp những QT có chi phí truyền tin cao hơn một ngưỡng C nào đó. Thêm vào đó, số các QT trong một nhóm đơn không thể vượt quá một ngưỡng X khác. Sử dụng ví dụ trong hình 5.7 và chi phí truyền tin trung bình được ước lượng C=9 như mộtngưỡng, 3 nhóm (2,4), (1,6), (3,5) có thể tìm ra. Hiển nhiên nhóm (2,4) và (1,6) phải được sắp đặt tương ứng cho các bộ xử lý A và B. Nhóm (3,5) có thể được ấn định cho bộ xử lý A hoặc B. Việc ấn định chúng cho B có chi phí tính toán thấp hơn nhưng phải chịu một chi phí truyền tin cao hơn nhiều. Vì vậy chúng được ấn định cho A, kết quả là chi phí tính toán trên A là 17, trên B là 14 và chi phí truyền tin giữa A và B là 10. Tổng chi phí là 41, một chi phí không cao hơn nhiều so với chi phí tối ưu là 38 nhận được từ thuật toán mặt cắt nhỏ nhất. Giá trị của ngưỡng X có thể được sử dụng để cân bằng sự thực hiện công việc trên các bộ xử lý. Sử dụng một giá trị X thích hợp để phân phối công việc cần làm thậm chí cũng sẽ ảnh hưởng tới sự phân chia (3,5) cho bộ xử lý A trong ví dụ tương tự.

Lịch trình tĩnh tối ưu có độ phức tạp cao. Các thuật toán đơn giản để tìm ra là hấp dẫn, thu hút. Mặc dù nhiều giải pháp để tìm ra tạo ra nhiều sự xét đoán nhưng chúng ta chỉ có những thông tin gần đúng về giá truyền thông và tình toán. Hơn thế nữa việc thi hành sự phân chia xử lý khởi tạo là không được phê bình nếu những xử lý có thể bị di chuyển sau khi chúng vừa được phân chia. Đó là một trong những thúc đẩy cho lập lịch xử lý động được đưa ra trong đoạn tiếp theo.

5.3 Chia xẻ và cân bằng động

Hai ví dụ về lập lịch của phần trên đây chính là cách thức lập lịch tĩnh. Khi một QT được đưa tới một nút, QT này được lưu lại đó cho đến khi nó được hoàn thiện. Cả 2 ví dụ trên đều đòi hỏi biết trước về thời gian chạy và cách thức truyền thông của quá trình. Với mô hình QT đi trước, mục tiêu đầu tiên là tối thiểu hoá thời gian hoàn thiện toàn bộ, trong khi mô hình QT TT cố gắng tối thiểu hoá tổng chi phí TT, đồng thời tìm cách thoả mãn những ràng buộc về tính toán. Một mô hình toán học và một thuật toán tốt là yếu tố cần thiết cho lập lịch. Tuy nhiên, việc tính toán lại tập trung và chỉ xảy ra tại một thời điểm định trước.

Biết trước thông tin về các QT là không thực tế trong hầu hết các ứng dụng phân tán. Với đòi hỏi kết nối và tính toán không cần thông tin trước, ta phải dựa trên một chiến lược lập lịch linh hoạt, cho phép những quyết định được thực hiện tại địa phương. Trong phần này, chúng ta sẽ sử dụng mô hình QT không liên kết để thể hiện một số chiến lược lập lịch động. Việc sử dụng mô hình không liên kết không có nghĩa là mọi QT không có liên hệ với nhau, mà được hiểu theo nghĩa: chúng ta không biết một QT này tương tác với các QT khác như thế nào. Vì vậy, ta có thể lập lịch với giả sử rằng chúng không kết nối. Điều này tương đương với việc bỏ qua sự phụ thuộc giữa các QT. Với mô hình này, mục tiêu của việc lập lịch khác so với mục tiêu của mô hình ưu tiên và mô hình liên hệ. Mục tiêu lớn nhất có thể thấy được trong lập lịch là hướng tới tính hiệu dụng (*utilzation)* của hệ thống và tính công bằng (*fairness)* cho các QT xử lý của người dùng. Tính hiệu dụng của các bộ xử lý có liên quan trực tiếp đến các thước đo tốc độ như khối lượng xử lý và thời gian hoàn thành. Sự công bằng rất khó để định nghĩa cũng như ảnh hưởng của nó đến hoạt động là không rõ ràng. Có thể nói hiệu dụng và công bằng là yêu cầu trong lập lịch cho mô hình không liên kết của một hệ thống phân tán.

Một chiến lược đơn giản để nâng cao hiệu quả sử dụng của một hệ thống là tránh được nhiều nhất tình trạng bộ xử lý rỗi. Giả sử rằng ta có thể chỉ định một QT điều khiển chứa đựng thông tin về kích thước hàng đợi của mỗi bộ xử lý. Các QT đến và ra khỏi

hệ thống theo phương thức dị bộ. Một *QT đến* sẽ đưa ra yêu cầu đòi hỏi bộ điều khiển cung cấp một bộ xử lý. Bộ điều khiển sẽ lập lịch điều phối đưa QT đó đến một *bộ xử lý* có hàng đợi ngắn nhất. Để cập nhập thông tin về kích thước hàng đợi, mỗi *bộ xử lý* cần cung cấp thông tin cho bộ điều khiển ngay khi một QT được hoàn tất và ra khỏi khu xử lý. Việc kết nối với hàng đợi ngắn nhất chính là chiến lược điều phối tĩnh cho chia xẻ nhiệm vụ (*static load sharing)* nhằm mục đích giảm bớt thời gian rỗi của các *bộ xử lý* và giảm sự chênh lệch về hàng đợi (cân đối nhiệm vụ) giữa các bộ xử lý. Việc cân đối tải là đòi hỏi cao hơn so với chia xẻ tải, bởi vì chúng nâng cao hiệu quả sử dụng và đưa tới một cách cân đối đúng theo nghĩa bằng nhau về nhiệm vụ phải thực hiện của mỗi bộ xử lý. Cân bằng nhiệm vụ có tác dụng làm giảm thời gian phí tổn trung bình của các QT. Chiến lược này có thể được sửa đổi bằng cách cho phép di chuyển linh động một QT từ hàng đợi dài đến các hàng đợi ngắn hơn. Mô hình hàng đợi trên đã được đề cập đến trong hình 5.3 c, mô hình trạm làm việc. Tính hiệu quả và cân bằng càng được nâng cao bởi phương thức phân phối linh động lại các công việc hay còn gọi di trú QT.

Tuy nhiên, sự cân bằng đề cập ở trên vẫn chưa mang thật đầy đủ ý nghĩa bởi nó dựa trên quan điểm của hệ thống hơn là của người dùng. Trong các QT được phát sinh bởi người dùng tại các trạm địa phương. Vì vậy, một hệ thống cân bằng theo quan điểm người sử dụng phải là một hệ thống ưu tiên cho chương trình của người dùng nếu chương trình đó đòi hỏi chia xẻ các tài nguyên tính toán ít hơn các chương trình khác. Trên nguyên tắc này, bộ điều khiển phải kiểm soát được *bộ xử lý* hiện đang cấp phát cho một QT của người sử dụng. Ngay khi một *bộ xử lý* rỗi, bộ điều khiển sẽ cấp phát *bộ xử lý* đó cho một QT đang chờ đợi tại phía có số lần được cấp phát CPU ít nhất. Tính hiệu dụng được thể hiện bằng cách cố gắng định vị tối đa các *bộ xử lý* có thể được. Tiêu chuẩn này có thể được điều chỉnh bằng việc tính toán độ dài hàng đợi, thông số phản ánh nhiệm vụ tại mỗi vùng và cũng vì thế thực hiện được sự cân bằng các QT được nạp. So sánh với phương pháp điều phối "hàng đợi kết nối với QT ngắn nhất" (*join-to-the-shortest queue),* ta có thể thấy phương pháp này cho một định nghĩa tốt hơn về sự công bằng, việc điều phối được khởi tạo bởi một QT tại điểm xuất phát thay vì tại điểm đích, và vì thế nó phù hợp hơn cho mô hình xâu-bộ xử lý.

Cuộc tranh luận quanh bất kỳ vấn đề nào về hệ phân tán sẽ không bao giờ kết thúc trừ phi ta chứng minh được tác dụng của sự điều khiển tập trung (hoặc chứng minh loại bỏ nó). Nếu chúng ta huỷ bỏ sự điều khiển tập trung trong việc chuyển giao một QT từ 1 vùng này (nơi gửi) đến 1 vùng khác (nơi nhận), công việc chuyển giao QT phải được tạo lập bởi nơi gửi, nơi nhận, hoặc cả hai. Trong 2 phần tiếp, chúng ta sẽ thảo luận Thuật toán tạo lập trạm gửi và thuật toán tạo lập từ trạm nhận cho công việc chuyển giao QT.

5.3.1. Thuật toán tạo lập từ trạm gửi

Thuật toán tạo lập từ trạm gửi mong muốn giảm bớt một phần nhiệm vụ tính toán. Thuật toán phân tán nhiệm vụ giúp chuyển các QT từ một trạm gửi có khối lượng công việc nặng tới nơi khối lượng công việc ít hơn được dễ dàng. Việc chuyển giao các QT đòi hỏi 3 chính sách cơ bản:

Chính sách chuyển nhượng: Khi nào một đỉnh trở thành trạm gửi?

Chính sách lựa chọn: Trạm gửi sẽ lựa chọn QT nào để gửi?

Chính sách định vị: Đỉnh nào sẽ là trạm nhận?

Khi khối lượng nhiệm vụ được thể hiện qua kích thước hàng đợi, trạm gửi có thể sử dụng chính sách chuyển nhượng (*transfer policy)* khi nhận thấy kích thước hàng đợi có thể vượt quá ngưỡng cho phép nếu nhận thêm một QT. Một QT mới đương nhiên là

ứng cử viên cho chính sách lựa chọn nếu không có lý do gì xoá bỏ nó. Với chính sách định vị thì khó khăn hơn bởi nó đòi hỏi một vài thông tin để định vị trạm nhận cho phù hợp. Trạm gửi cũng có thể lựa chọn ngẫu nhiên các đỉnh thuận. Tuy nhiên, việc này sẽ gây ra một chuỗi thao tác chuyển nhượng QT nếu đỉnh được chọn lựa lại bị quá tải. Trừ phi có một số thông tin tổng thể về tình trạng phân bố công việc, nếu không nơi gửi bắt buộc phải thăm dò đơn giản là xét thử một số giới hạn số trong một lần, chọn đỉnh có hàng đợi ngắn nhất làm nơi nhận, với điều kiện độ dài hàng đợi nơi nhận sẽ nhỏ hơn hoặc bằng độ dài hàng đợi nơi gửi sau khi chuyểen nhượng QT. Tất nhiên, QT thăm dò có thể dừng sớm hơn nếu một đỉnh rỗi được tìm ra trước khi đạt tới giới hạn thăm dò.

Sự thăm dò các đỉnh nhận và công việc chuyển giao các QT giữa nơi gửi và nơi nhận cần tính tới chi phí kết nối, một nguyên nhân tăng thời gian nạp chương trình thực tế của hệ thống. Trong một hệ thực sự tải nặng, vấn đề trên có thể còn tồi tệ hơn bởi ảnh hưởng của hiệu ứng *ping-pong (QT bị chuyển trên mạng liên tục)*, các trạm gửi cố gắng giảm nhẹ nhiệm vụ một cách vô ích, bởi mọi đỉnh đều có thuật toán tạo lập như nhau.

Tuy nhiên, thuật toán tạo lập từ trạm gửi hoạt động rất tốt khi hệ tải nhẹ. Với mức tải không nặng lắp, ta dễ dàng rìm ra được nơi nhận, phí tổn kết nối là không đáng kể. Một trong những hướng cải tiến đang được nghiên cứu là chọn lựa **ST** và **PL** phù hợp với các chiến lược thăm dò khác nhau.

5.3.2. Thuật toán tạo lập từ trạm nhận

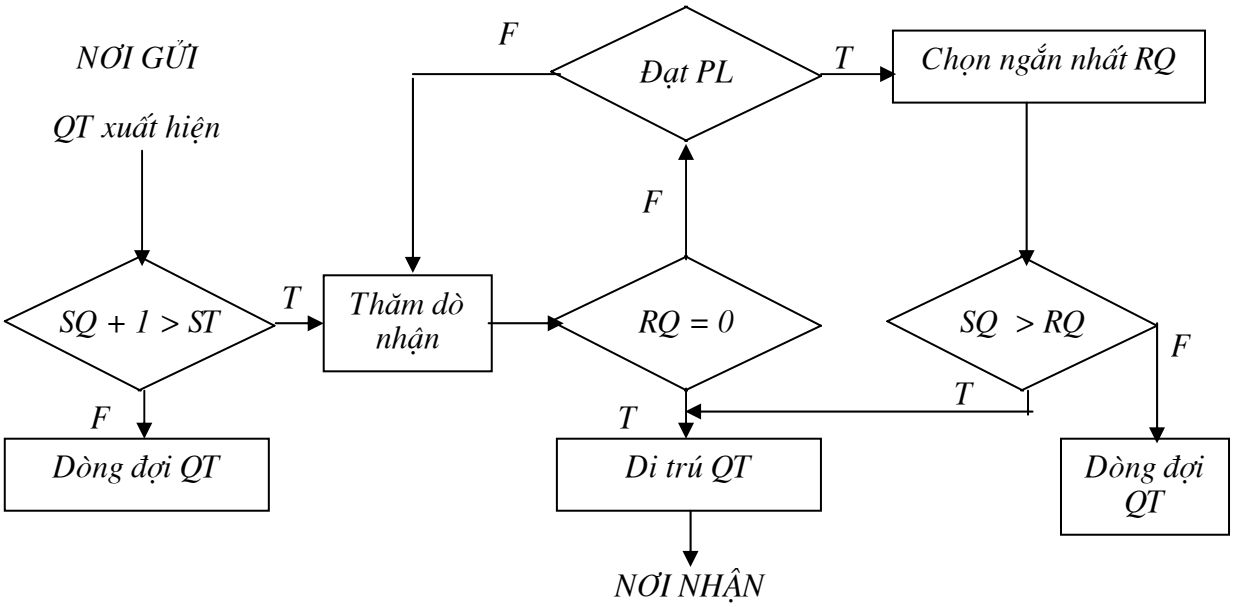
Như đã thấy ở trên, thuật toán phân chia nhiệm vụ tạo lập từ trạm gửi giống như một mô hình "đẩy", trong đó 1 QT được đẩy từ một bộ xử lý này tới bộ xử lý khác. Tương ứng với nó, một đỉnh nhận có thể kéo một QT từ một *bộ xử lý* khác về để xử lý: thuật toán lập tạo từ trạm nhận. Sử dụng *chính sách chuyển nhượng* tương tự như trên, thuật toán này sẽ tạo lập thao tác "kéo" khi độ dài hàng đợi tụt xuống dưới một ngưỡng RT (đã được định trước) vào thời điểm bắt đầu một QT. Một chiến lược thăm dò tương tự cũng được sử dụng trong *chính sách định vị* để tìm kiếm một đỉnh gửi đã quá tải. Tuy nhiên, *chính sách lựa chọn* lại đòi hỏi một thứ tự ưu tiên khi các QT tại trạm gửi đã bắt đầu chạy. Việc quyết định QT nào chuyển đi sẽ không rõ ràng như trong thuật toán tạo lập từ trạm gửi. Ta phải tính sao cho lợi ích thu được từ việc chia xẻ nhiệm vụ phải lớn hơn phí tổn tính độ ưu tiên và phí tổn cho liên lạc.

Thuật toán tạo lập từ trạm nhận có tính ổn định hơn thuật toán tạo lập từ trạm gửi. Trong một hệ thống có mức tải lớn, việc di chuyển các QT xẩy ra ít, các trạm gửi được tìm thấy dễ dàng, lượng công việc được chia xẻ hiệu quả, phí tổn ít. Khi mức tải của hệ thống ở mức thấp, việc tạo lập các di chuyển xảy ra nhiều nhưng vẫn không làm giảm hoạt động của thuật toán. Tính trung bình, thuật toán tạo lập từ trạm nhận hoạt động tốt hơn thuật toán tạo lập từ trạm gửi.

Điều tất yếu là tìm cách kết hợp hai thuật toán. Ví dụ, một trạm xử lý có thể sử dụng thuật toán tạo lập từ trạm gửi khi hàng đợi qua ngưỡng giới hạn ST cũng như có thể kích hoạt thuật toán tạo lập từ trạm nhận khi kích cỡ hàng đợi giảm thiểu xuống dưới ngưỡng RT. Việc lựa chọn giữa 2 thuật toán dựa trên thông tin đánh giá về mức tải của hệ thống. Nếu 2 thuật toán trên là đối xứng và không linh hoạt thì việc kết hợp nói trên chính là một thuật toán thích ứng. Trong cả hai trường hợp (tải nặng hoặc nhẹ), mỗi trạm có thể linh hoạt đóng vai trò của trạm nhận hoặc trạm gửi. Các trạm gửi sẽ gặp trạm nhận tại các *điểm hẹn*.

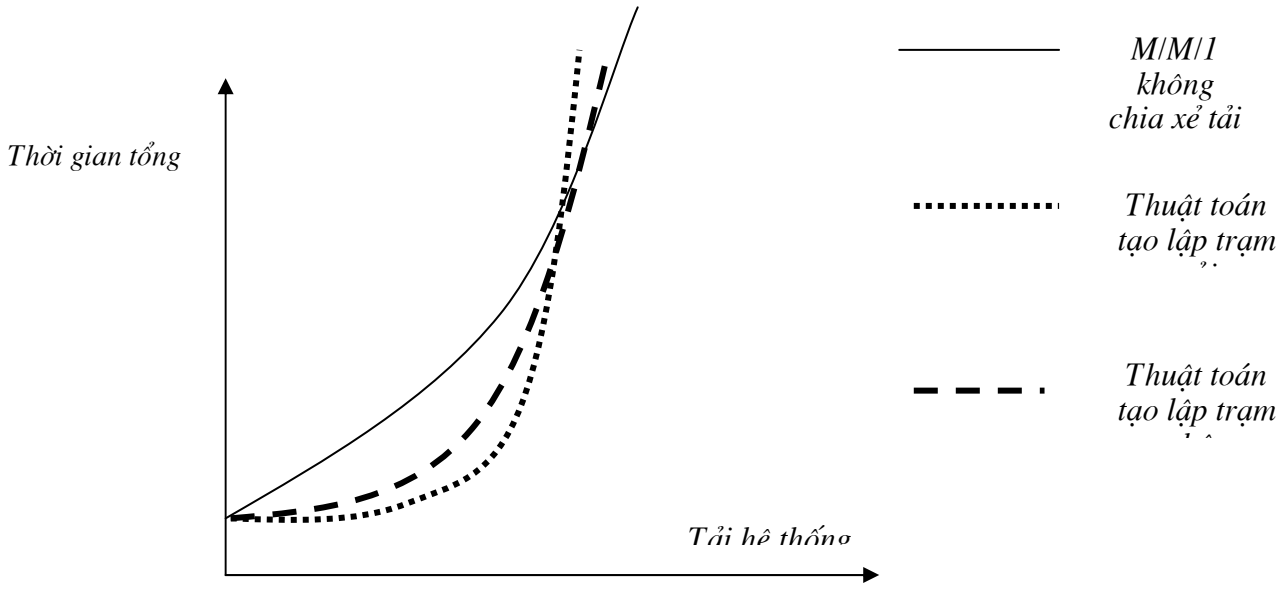
Để tạo lập trên thực tế các điểm hẹn này, một dịch vụ đăng ký (*registration service)* được dùng để kết hợp 1 trạm gửi với một tạm nhận. Việc thăm dò vì thế mà trở thành không cần thiết. Trạm phục vụ đăng ký(regisration phục vụ) hoạt động như một " thương nhân" trao đổi giữa người trả giá cao nhất (sender) với người cung cấp rẻ nhất

(receiver) mà giá cả hàng hoá thời gian thực hiện các QT. Một trạm " tốt" phải biết dùng thuật toán tạo lập từ trạm gửi, kích hoạt thuật toán tạo lập từ trạm nhận khi trạm cảm thấy hệ thống tải ở mức cao, và hoạt động ngượclại khi mức tải là thấp.Thuật toán vì thế sẽ tương thích với sợ thay đổi của hệ thống.



5.9. Sơ đồ khối thuật toán tạo lập từ trạm nhận

Hình 5. 10 so sánh hoạt động của thuật toán linh hoạt chia sẻ công việc. Thời gian lãng phí của hệ thống M/M/1 không chia xẻ tải là đường cơ sở cho việc so sánh.



Hình 5.10. So sánh hoạt động của các thuật toán chia sẻ công việc động

5.4 Thi hành quá trình phân tán

Chiến lược chia sẻ tải tĩnh hay động đều đòi hỏi thực hiện QT trên một trạm xa. Việc tạo lập một QT từ xa có thể được thực thi bằng mô hình Client/Server), tương tự như cách thực thi của RPC. Trên hình 5.11 giả sử đã có các QT nền điểm-vào giúp cho việc tạo lập và kết nối các QT trên các máy khác nhau được dễ dạng. Một QT cục bộ trên

một máy Khách trước hết cần tạo một yêu cầu tới các *QT xử lý đầu cuối*, các QT này có liên hệ với những "nền"(*stub)* nằm trên phục vụ đại diện cho QT đó. Nếu yêu cầu này được chấp nhận và mọi tài nguyên cần thiết đều được đáp ứng, "nền" trên phục vụ. Mọi liên lạc tiếp theo giữa địa phương và QT ở xa sẽ được giúp đỡ gián tiếp thông qua các QT nền. Các QT cơ sở phục vụ như một kết nối logic, tạo lập ranh giới vật lý giữa QT địa phương và QT ở xa.

Dựa trên cách thức phiên dịch một thông điệp yêu cầu, có 3 thể loại ứng dụng chính:

Dịch vụ từ xa (remote service): Thông điệp được hiểu như một yêu cầu cho một *service* đã biết tại một trạm xa.

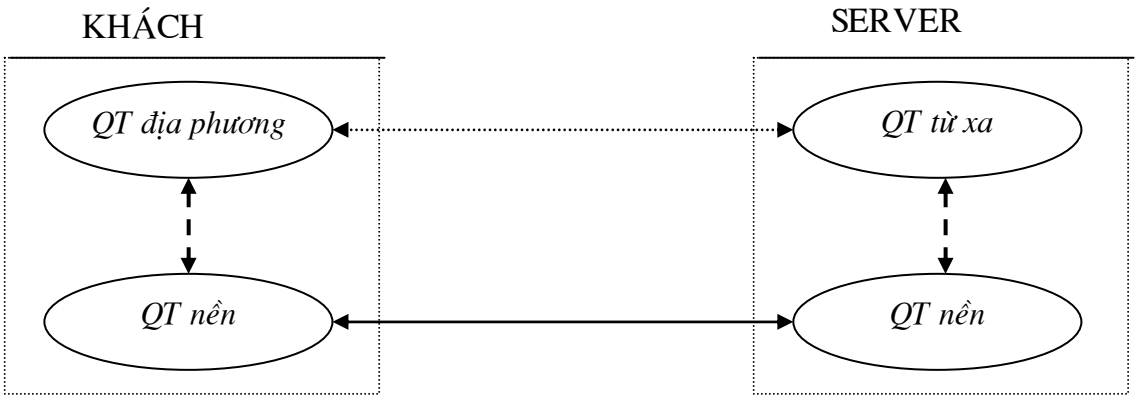
Thực hiện từ xa (Remoce execution): Thông điệp chứa đựng một chương trình sẽ được thực hiện tại một *remote site*.

Di trú QT: Thông điệp đại diện cho một QT đang được chuyển đến một *remote site* để tiếp tục thực hiện.

Mỗi ứng dụng đòi hỏi phải có các biện pháp xử lý khác nhau được trình bày dưới đây.

5.4.1. Phục vụ từ xa

Remote service là một định nghĩa quen thuộc. Những ứng dụng đầu tiên của dịch vụ này là sự chia xẻ tài nguyên trong hệ thống phân tán. Với sự cho phép truy cập từ



Hình 5.11. Mô hình lôgic của QT cục bộ và từ xa

xa, nhiều Khách trên các máy khác nhau có thể cùng chia xẻ tài nguyên chung như: file hệ thống, thiết bị ngoại vi… Một thông điệp yêu cầu *dịch vụ từ xa* có thể được phân thành 3 mức phần mềm khác nhau:

Lời gọi thủ tục từ xa: mức ngôn ngữ.

Lệnh gọi từ xa (remote commands): mức HĐH

Thông điệp biên dịch (intepretive messages): mức trình ứng dụng.

Tại mức ngôn ngữ, RPC được coi như là mô hình thích hợp nhất cho các yêu cầu dịch vụ từ xa. Đó là loại hình hướng dịch vụ, cung cấp sự truy cập trong suốt cũng như định vị trong suốt (công việc được thực hiện trên máy chủ, người dùng không nhìn thấy).

Tại mức HĐH, có một số lệnh thường xuyên được các đối tượng từ xa sử dụng. Những lệnh này được gắn liền thành một phần của 1 lệnh khung (*shell command)* và được HĐH địa phương chấp nhận. Ví dụ lênh *rcp* trong UNIX, lệnh coppy một file từ xa, rất hay sử dụng. Điều này có thể mở rộng cho các lệnh khác bằng việc tạo một lệnh khung cho phép người dùng chạy một lệnh khung tại bất kỳ 1 hệ thống từ xa. Ví dụ lệnh *rsh host-l user ls* trong UNIX dùng để liệt kê các files trên trang chủ của người dùng,

User, trên máy chủ, *Host*. Như vậy *Rsh* là một lệnh xa (*remote command)*. Ta có thể phát triển bằng cách đưa *rsd* vào trong một file lệnh (*script file)*, cho phép thực thi nhiều lệnh trong 1 lần gọi (*giống . bat)*. Ngày nay *remote command* đơn giản có mặt hầu hết trên các máy mới nhằm phục vụ cho nối mạng.

Lệnh từ xa bị giới hạn ở những lệnh shell. Ý tưởng trên có thể được mở rộng để xử lý các thông điệp. Một người dùng có thể gửi 1 thông điệp tới 1 máy chủ yêu cầu một số thao tác do người dùng định nghĩa trong nội dung thông điệp. Nó giống như một RPC tại mức hệ thống. Trong trường hợp này, QT nền tại nơi phục vụ phải có chức năng biên dịch các thông điệp gửi từ bộ xử lý cơ sở trên Khách và có các thao tác tương ứng với yêu cầu. Nguyên tắc quản lý việc truyền và xử lý thông điệp trở thành một giao thức truyền thông ứng dụng (*Application communication protocol)* giữa Khách và phục vụ. Một ví dụ điển hình là giao thức truyền Phục vụ file cho fpt. Chúng biên dịch các lệnh như *get, put* thành các thao tác *downloading* và *uploading* tương ứng. Sử dụng quá trình daemon là một kỹ thuật phổ biến trong lập trình mạng.

Các thao tác xa được khởi xướng qua RPC, lệnh xa và thông điệp thông dịch (*interpretive message)* chỉ là những phục vụ mà máy chủ cung cấp. Vấn đề đầu tiên của mọi hoạt động là chuyển hướng vào/ra và an ninh. Với việc chuyển hướng, *khách stb* copy các dữ liệu vào chuẩn *của QT người dùng* cho các lệnh xa và nền phục vụ trả lại các kết quả chuẩn, các lỗi sinh ra của lệnh đó có cho chương trình người dùng.

5.4.2. Thực hiên từ xa

Thực hiện từ xa khác dịch vụ từ xa ở chỗ: một thao tác từ xa *(remote operation)* được đề ra và kiến tạo bởi chính Khách trong khi đó tại mức dịch vụ từ xa, Khách chỉ đề ra thao tác, còn các thao tác này đã được tạo sẵn trên phục vụ. Một thông điệp gửi đi từ Khách chính là chương trình của Khách dùng để chạy trên máy chủ. Một máy chủ có thể là một hệ thống có tài nguyên đặc biệt hoặc đơn giản đó là bất kỳ một hệ thống nào dùng cho mục đích chia xẻ công việc. Hệ thống có tài nguyên đặc biệt chính là trường hợp chung của dịch vụ từ xa. Phần còn lại chính là mô hình xâu-*bộ xử lý* dùng cho những hoạt động phân tán (*Thực hiện từ xa)* hoặc *định vị động các bài toán (dynamic task placement).*

Sự khác biệt lớn nhất giữa dịch vụ từ xa và thực hiện từ xa là môi trường hoạt động. Do mục đích của dịch vụ từ xa là truy cập các tài nguyên ở xa, vì vậy, mọi điều cần biết về các QT xử lý từ xa đều nằm ở máy chủ. Trái lại, với *thực hiện từ xa,* các QT xử lý xa chứa đựng các thông tin về hệ thống gốc. Các máy chủ chỉ đơn giản làm nhiệm vụ giảm nhẹ công việc tính toán. Độ phức tạp của việc thực thi các Thực hiện từ xa tăng lên đáng kể khi nhiều QT ở xa có ảnh hưởng lẫn nhau được tạo ra đồng thời. Các vấn đề nảy sinh là:

Thuật toán phân chia công việc

Đơn vị độc lập

Tính không đồng nhất của hệ thống

Bảo mật và an toàn.

Để đơn giản hoá, ta giả sử rằng một dịch vụ QT tồn tại trên mọi máy. Dịch vụ QT có trách nhiệm lưu giữ những thông tin về công việc, thoả thuận với máy chủ, gọi các thao tác từ xa, tạo lập các QT nền để kết nối Khách và phục vụ. Thực hiện từ xa có thể được khởi xướng một cách rõ ràng bởi một QT (có thể hoàn toàn từ một QT xử lý trên phục vụ QT địa phương. Vì vậy, mối liên hệ giữa các QT có thể là quan hệ cha – con hoặc quan hệ không liên kết *(disjont relation ship or noninteracting)*. Trong cả 2 trường hợp, công việc đầu tiên vẫn là chọn máy chủ ở xa. Tuỳ theo các QT trên máy

chủ mà thuật toán tạo lập từ trạm gửi hoặc thuật toán tạo lập từ trạm nhận sẽ được áp dụng. Trong thực tế, mỗi QT xử lý lưu giữ một danh sách các máy chủ đã đăng ký và đang sẵn sàng đảm nhận một *thực hiện từ xa*. QT đăng ký/huỷ bỏ thực hiện thông qua việc quảng bá. QT lựa chọn phục vụ được thực hiện thông qua một QT môi giới tập trung. Sau khi chọn trạm xa, QT thương lượng bắt đầu. Phục vụ QT Khách thông báo cho phục vụ QT tại trạm xa yêu cầu về các tài nguyên. Nếu các tài nguyên yêu cầu chấp nhận và Khách được xác nhận, phục vụ sẽ cho phép thực thi Thực hiện từ xa. Việc truyền mã chương trình được thực hiện, sau đó phục vụ tạo lập các QT từ xa và tạo lập nền. Cuối cùng, Khách khởi động QT đã được phân chia cho trạm xa đó.

Tính độc lập định vị trong *thực hiện từ xa* có đòi hỏi cao hơn so với định hướng lại vào/ra trong dịch vụ từ xa. Các QT tạo lập bởi Thực hiện từ xa đòi hỏi sự phối hợp để hoàn thành công việc chung. Vì thế càn cung cấp cho mỗi QT một thông tin tổng thể cho dù chúng đều đang chạy trên các máy đơn. Mỗi QT xa có một đại diện nằm trên máy chủ đầu tiên. Quan hệ cha/con được thiết lập. Mọi kỹ thuật giao tiếp đa xử lý được thực hiện trong suốt đinh vị. Các file hệ thống của máy chủ đầu tiên thường xuyên cung cấp thông tin tổng thể cho các QT.

Thông thường, thực hiện từ xa thực hiện trên một môi trường đồng nhất trong đó các máy tính tương thích cả về phần cứng cũng như phần mềm. Khi một Thực hiện từ xa được gọi trên một máy chủ không tương thích, chương trình cần phản linh dịch lại, và phí tổn nhiều khi là quá cao. Một giải pháp cho vấn đề này là sử dụng ngôn ngữ trung gian độc lập (*canonical machine-independent intermediate language)* để lập trình từ xa, ví dụ như Java. Chương trình ghi trên Java được linh dịch thành bộ mã độc lập. Bộ mã này có thể linh dịch trên mọi máy chủ có trang bị bộ dịch mã *bytecodes*. Các đối tượng trên mạng được đánh địa chỉ duy nhất trong chương trình Java thông qua bộ định vị tài nguyên tổng thể. Cùng với vấn đề mã tương thích, việc trao đổi dữ liệu giữa các vùng không đồng nhất cũng cần phải giải quyết, thông tin cần được chuyển đổi. Một lần nữa, việc sử dụng dữ liệu tổng thể (*ví dụ, XDR_external data representation)* cần được tích hợp vào các phương tiện cơ bản của Thực hiện từ xa.

Tuy nhiên, *Thực hiện từ xa* có hai mặt của nó. Nó có đầy đủ sức mạnh nhưng lại đem lại sự lạm dụng hệ thống. Một mã chương trình lạ có thể làm hại chính người dùng. Vì thế, trên quan điểm về bảo mật và an toàn, sẽ là đáng tin cậy hơn khi chỉ chấp nhận duy nhất các thực hiện từ xa có mã gốc hoặc bộ mã trung gian. Ngôn ngữ dùng để lập trình một thực hiện từ xa nên được giới hạn để loại trừ các khả năng xấu có thể xảy ra (*ví dụ: con trỏ và đa thừa kế (pointer & multiple inheritance)*. Trong trường hợp một bộ mã trung gian được sử dụng, ta bắt buộc phải kiểm tra để đảm bảo chắc chắn mã này được sinh ra từ một chương trình nguồn thực sự. Kiểm tra tham số trong khi chạy, kiểm tra tràn Stack cũng rất cần thiết để bảo vệ sự toàn vẹn của các trạm xa. Do đó, vấn đề bảo mật và an toàn cho các Thực hiện từ xa của hệ thống phân tán vẫn là chủ đề đang được nghiên cứu.

5.4.3. Di trú quá trình

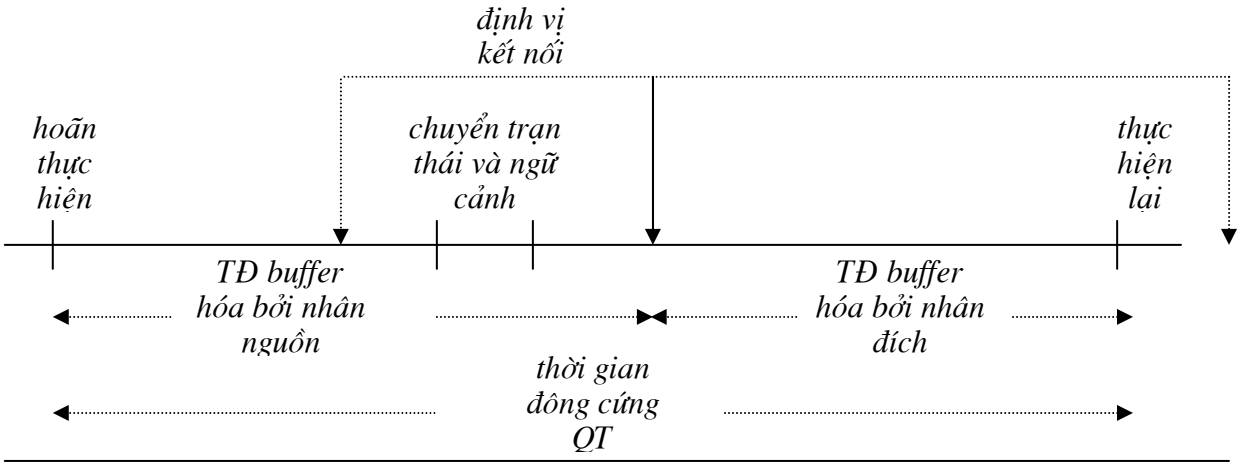
Trong vấn đề thực hiện từ xa nêu ở trên, một thao tác khi đã bắt đầu sẽ tồn tại trên trạm cho đến khi hoàn thành. Chúng ta có thể mở rộng mô hình chia xẻ tải cho phép một Thực hiện từ xa có thể giành quyền chuyển sang một trạm khác. Như vậy, một QT có thể di chuyển linh hoạt từ trạm này tới trạm khác. Sự di chuyển các QT là một chủ đề rất hấp dẫn. Một hệ thống với năng lực trong suốt di trú là thành quả cuối cùng của xử lý phân tán.

Cũng giống như Thực hiện từ xa, một chức năng di chuyển QT đòi hỏi phải định vị và thương lượng được với 1 trạm xa, chuyển nhượng mã, khởi động hoạt động. Và khi

một QT được di chuyển, các trạng thái của nó cũng phải chuyển kem theo. Trạng thái của một QT trong hệ phân tán bao gồm 2 phần: trạng thái tính toán và trạng thái truyền thông. Trạng thái tính toán là những thông tin cần thiết để lưu và thiết lập lại một QT trên một trạm xa. Trạng thái truyền thônglà tình trạng của các mối liên kết và các thông điệp quá cảnh (*các thông điệp đang tạm thời lưu giữ chờ chuyển tiếp).* Việc chuyển nhượng trạng thái kết nối là một vấn đề mới trong thực thi việc di chuyển QT.

Định hướng lại liên kết và chuyển phát thông điệp

Các QT dùng các liên kết truyền thông cho mục đích liên lạc giữa các QT. Chúng được thực hiện thông qua bảng liên kết (*link table)* chứa trong nhân. Bảng liên kết chứa các con trỏ trỏ tới điểm kết nối cuối (*communication endpoints)* của các QT xử lý khác liên quan đến nó. Khi di chuyển một QT, bảng liên kết (của QT có mối liên hệ với QT được di chuyển) cần được cập nhật lại để giữ nguyên được các mối liên kết đã có. Rất nhiều giải pháp cho máy tính được tìm thấy trong đời sống hàng ngày. Việc chuyển hướng liên kết cũng giống như việc chuyển địa chỉ khi ta thay đổi nơi sinh sống. Thông thường, ta sẽ thông báo địa chỉ mới cho các bạn thân trước khi di chuyển và cho những người còn lại sau khi chuyển. Cũng với phương thức như vậy, việc chuyển hướng liên kết được thực hiện như 1 trong những công đoạn của việc di chuyển QT, trước hoặc sau khi chuyển các ngữ cảnh, như được trình bày trên hình 5.12. Đầu tiên QT di chuyển sẽ ngưng lại (*subpended or frozen)* ngay sau khi lựa chọn và thương lượng được với một trạm xa. Và khi trạm xa đã sẵn sàng, công việc chính tiếp theo là chuyển giao hiện trạng và ngữ cảnh của chương trình (*chuyển bản mã chương trình)* tới trạm xa trước khi công việc được thực hiện lại tại đây. Việc chuyển hướng liên kết có thể được thực hiện bằng cách gửi một yêu cầu cập nhật liên kết cho các QT có liên



Hình 5.12. Định hướng lại kết nối và chuyển tiếp TĐ

quan. Thời gian cho cập nhật liên kết ảnh hưởng đến việc các thông điệp gửi đến trong QT di chuyển được chuyển tiếp như thế nào. Những thông điệp gửi đến trước khi cập nhật liên kết được lưu giữ, có thể được chuyển đồng thời với mã nguồn (hoặc chuyển muộn hơn thông qua nhân nguồn (*source kernel)-phần chương trình cốt yếu còn lại ở trạm cũ*. Sau khi cập nhật liên kết, các thông điệp phải đếm trước khi chương trình hoạt động trở lại trên trạm mới. Chúng được chứa trong *bufers* bởi nhân đích (*destintation kernel) –phần chương trình cốt yếu nằm trên trạm mới*. Thực hiện cập nhật liên kết sớm sẽ giảm bớt công việc thừa do phải lưu thông điệp tại nhân nguồn. Một cách lý tưởng, mọi thứ còn lại tại trạm gốc sau QT di chuyển là nhỏ nhất và được dọn gọn nhanh nhất có thể. Ngược lại, nó sẽ làm hỏng mục đích giảm nhẹ công việc.

Tuy nhiên, ngay cả khi việc cập nhật diễn ra nhanh chóng, sau khi QT được di chuyển, các thông điệp vẫn có thể đến trạm cũ do sự trễ trên mạng hoặc do nơi gửi không biết gì về việc di chuyển. Để không mất thông tin, nhân nguồn cần phải tiếp tục chuyển những thông điệp tới QT đã được di chuyển. Theo lý thuyết, quãng thời gian này là không xác định. Trên thực tế, ta cần đặt ra một giới hạn giống như hạn gửi thư trên bưu điện. Trong khi chưa hết hạn, các thông điệp được chuyển giao cho nhân đích. Để giảm bớt sự truyền không trực tiếp, nhân nguồn sẽ thông tin cho nơi gửi vị trí mới của QT. Nhưng việc thông báo này chỉ thực hiện được khi nhân nguồn biết được thông tin về nơi gửi. Những thông điệp đến sau thời gian cho phép sẽ bị bỏ qua và coi như thất lạc. Vì vậy, chương trình ứng dụng phải có trách nhiệm xử lý thông tin bị thất lạc.

Chuyển giao ngữ cảnh và trạng thái

Thời gian từ khi dừng chương trình đến khi tái hoạt động của một QT gọi là thời gian đông cứng. Đó là cái giá phải trả cho việc di chuyển các QT. Để giảm bớt phí tổn, các QT chuyển ngữ cảnh (*context transfer)*, chuyển hướng liên kết (*link redirection)*, chuyển phát thông điệp cần phải xử lý đồng thời. Trong thực tế, việc chuyển hướng liên kết và chuyển phát thông điệp có thể đợi khi QT được tái hoạt động ở địa điểm mới. Điều kiện duy nhất cần thiết cho một QT có thể định danh hoạt động ở địa điểm mới là sự giao giao tình trạng hoạt động và một vài mã khởi tạo. Như vậy, để giảm bớt thời gian đông cứng, điểm tái hoạt động (*resume execution)* của QT trên hình 5.12 cần được đẩy lùi và gối lên QT chuyển ngữ cảnh. Nếu bản mã lớn, QT chuyển có thể được thực hiện theo gói các khối hoặc theo trang. Mã khởi tạo được chuyển tới, thậm chí trước khi QT di chuyển được hoàn thành. Những khối mã khác có thể được copy theo chỉ dẫn: giống như hệ thống đòi hỏi trang. Mặc dù giảm được đáng kể thời gian đông cứng, nhưng phương pháp lại phụ thuộc vào việc tính toán trên trạm nguồn. Tuy nhiên, phương pháp này tỏ ra rất phù hợp với hệ thống chia xẻ bộ nhớ phân tán được nói tới ở chương 7. Một *hệ thống chia xẻ bộ nhớ phân tán* giả lập một bộ nhớ logic chung dựa trên các modul bộ nhớ vật lý phân tán. Vị trí của các khối bộ nhớ vật lý, được bản đồ hoá thành không gian địa chỉ nhớ logic của các QT, là trong suốt đối với các QT. Trong hệ thống như vậy, chỉ có thông tin trạng thái là cần chuyển giao. Việc chuyển giao ngữ cảnh là không cần thiết. Nó được ẩn giấu trong các kỹ thuật cơ sở làm nhiệm vụ chia sẻ bộ nhớ phân tán. Việc quyết định khi nào các khối, do QT đòi hỏi, được copy (*thậm chí định danh lại)* là trong suốt đối với QT. Những phụ thuộc vô ích không còn nữa. Vì vậy, nó nâng cao tốc độ truyền thông tin dẫn tới đẩy mạnh tốc độ chương trình.

5.5. Lập lịch thời gian thực

Lập lịch QT có nhiều dạng khác nhau khi thêm vào ràng buộc thời gian. Trong nhiều ứng dụng, HĐH cần đảm bảo việc sắp xếp các thao tác sao cho chúng tuân thủ các ràng buộc thời gian đã đặc tả. Hệ thống này được gọi là *hệ thống thời gian thực* vì chúng có ràng buộc tới hạn thời gian thực. Tồn tại nhiều hệ thống máy tính thời gian thực, như hệ thống máy tính hàng không, máy tính điều khiển tự động hoá, hệ tự động hóa sản xuất, hệ thống thương mại chứng khoán.

Dịch vụ thời gian thực được gắn với tập các tác vụ thời gian thực. Mỗi tác vụ τ được miêu tả bằng:

$$\tau_i = (S_i, C_i, D_i)$$

trong đó S_i là thời điểm sớm nhất có thể bắt đầu tác vụ τ_i, C_i là thời gian thực hiện trong trường hợp xấu nhất của τ_i và D_i là thời điểm chết của τ_i. Tập V tác vụ thời gian thực là:

$V = \{ \tau_i / i=1 \ldots n \}$

Tồn tại các dạng hệ thời gian thực chủ yếu sau đây

• Trong các hệ thống thời gian thực liên quan tới điều khiển theo mức độ an toàn hoặc thiết bị có mức tới hạn, mọi tác vụ buộc phải hoàn thành trước điểm chết nếu không tai hoạ xảy ra. Hệ thống này gọi là *hệ thống thời gian thực cứng* vì chỉ được coi là đúng nếu như mọi tác vụ phải bảo đảm hoàn thành trước điểm chết.

• Trong các hệ thống (như hệ thống đa phương tiện) có những điểm chết nhưng vẫn có thể hoạt động nếu không lỡ qua điểm chết quá nhiều lần. Chúng được gọi là *hệ thống thời gian thực mềm*. Ở đây một tác vụ vẫn tiếp tục phải hoàn thành dẫu rằng đã qua điểm chết.

• *Hệ thống thời gian thực tĩnh* về mặt nào đó tương tự như hệ thống thời gian thực mềm nhưng tác vụ qua điểm chết sẽ không được thực hiện. *Ví dụ*, máy tính tự động sản xuất không thể khởi sự một thao tác cơ học để tránh cho thiết bị khỏi tự làm hư hỏng.

• Nếu tác vụ được xuất hiện theo cách tại thời điểm tuỳ ý, chúng được gọi là *không theo chu kì*. Ở nhiều hệ thống thời gian thực thì thời điểm tác vụ xảy ra, khoảng thời gian thực hiện và điểm chết có thể tiên đoán được. Tập tác vụ như vậy gọi là *có chu kì*.

Ví dụ, máy tính điều khiển động cơ phải tính được lượng nhiên liệu và suy ra thời gian tối đa động cơ còn vận hành liên tục trước khi nạp nhiên liệu.

Miêu tả tập tác vụ thời gian thực có chu kì đơn giản. Mỗi tác vụ thực hiện một trong n công việc. Yêu cầu thực hiện công việc i một lần trong khoảng T_i giây. Các phần công việc phía trước phải hoàn thành trước khi phần công việc mới được bắt đầu. Như vậy, thời điểm bắt đầu của tác vụ mới là điểm chết của tác vụ cũ. Do vậy tại một thời điểm chỉ một tác vụ được thực hiện, ta gán cho tác vụ thực hiện phần công việc i là τ_i. Đặc tả tập tác vụ rút gọn lại bởi tập thời khoảng và khoảng thời gian thực hiện n công việc:

$V = \{J_i = (C_i,T_i) / 1 \le i \le n\}$

Ở đây chỉ quan tâm đến lập lịch các tác vụ theo đó, tác vụ thỏa mãn ràng buộc về thời điểm chết. Chúng ta cũng chỉ quan tâm lập lịch trên hệ thống đơn xử lý. Lịch là phân công CPU cho các tác vụ thời gian thực mà nhiều nhất một tác vụ được phân công tại thời điểm bất kỳ. Chính xác hơn, lịch là tập A các khoảng thời gian được miêu tả:

$A = \{(s_i,f_i,t_i) / i=1 \ldots n\}$

Trong đó s_i là thời điểm bắt đầu, f_i là thời điểm kết thúc và t_i là tác vụ được thực hiện trong khoảng thời gian đó. Lịch có giá trị chỉ khi thoả mãn các ràng buộc:

1. $\forall i=1, \ldots, \quad s_i < f_i$
2. $\forall i=1, \ldots, \quad f_i < s_{i+1}$
3. Nếu $t_i=k$ thì $S_k \le s_i$ và $f_i \le D_k$

Điều kiện 1 đòi hỏi khoảng thời gian thực hiện thực sự là một khoảng. Điều kiện 2 đòi hỏi các khoảng thời gian được sắp theo thứ tự. Điều kiện 3 đòi hỏi tác vụ phải thực hiện sau thời điểm cho phép và phải hoàn thành trước điểm chết. Tập tác vụ được gọi là khả thi nếu mọi tác vụ τ_k nhận được tối thiết C_k giây CPU thực hiện trong lịch. Tức là, nếu gọi:

$A(\tau_k) = \{a=(s_i,f_i,t_i) / a \in A \text{ và } t_i=k\}$

Như vậy, lịch là *khả thi* nếu mọi $\tau_k \in V$ mà

$$\sum_{(s_i, f_i, k=\tau_i)} f_i - s_i \geq C_k$$

Tập tác vụ được gọi là *khả thi* nếu tồn tại một lập lịch khả thi cho tập tác vụ đó. Mục đích của thuật toán lập lịch thời gian thực là tìm lịch khả thi nếu nó tồn tại. Chương này tìm hiểu các thuật toán lập lịch thời gian thực với giả thuyết ràng buộc khác nhau. Để đơn giản, chỉ xem xét hệ thống thời gian thực cứng.

5.5.1. Hệ thống đơn điệu đều

Kiểu đơn giản nhất của lập lịch thời gian thực có các giả thiết sau:

1. Mọi tác vụ là chu kỳ và T_i là khoảng thời gian của tác vụ τ_i
2. Các tác vụ không truyền thông tới nhau
3. Các tác vụ có mức độ ưu tiên và độ ưu tiên đó là cố định (lập lịch ưu tiên tĩnh).

Việc cố định các độ ưu tiên đã làm đơn giản hoá bài toán lập lịch. Trong lập lịch ưu tiên tĩnh, bộ lập lịch chỉ tìm kiếm tác vụ có độ ưu tiên cao nhất cho bước kế tiếp. Trong khi đó ở hệ độ ưu tiên động, bộ lập lịch phải tính toán lại giá trị ưu tiên sau các bước. Tuy nhiên, dẫu có nhiều tập tác vụ thích hợp với cách ưu tiên động, nhưng lớp bài toán lập lịch ưu tiên tĩnh vẫn là ý tưởng khởi đầu tốt và dễ thực hiện.

Quan sát tập tác vụ khả thi đơn giản chưa được lập lịch ưu tiên tĩnh. Nếu tập tác vụ đó có một lịch khả thi được tạo ra bằng bộ lập lịch ưu tiên tĩnh thì gọi tập tác vụ như thế là *có một lịch phân công ưu tiên tĩnh khả thi*. Mục đích của phần này là tìm lịch trên nếu nó tồn tại.

Nếu tác vụ τ_i được yêu cầu thực hiện tại thời điểm t, τ_i sẽ không vượt điểm chết nếu thời gian thực hiện các tác vụ ưu tiên cao hơn trong khoảng thời gian $(t,t+D_i)$ nhỏ hơn hay bằng D_i-C_i. Khi thực hiện tác vụ được đòi hỏi (tức là, bắt đầu thời khoảng của nó), tác vụ có thể hoặc không thể hoàn thành công việc trước điểm chết, phụ thuộc vào dịch vụ CPU đòi hỏi cho các tác vụ ưu tiên cao hơn. Tuy nhiên, có thể định vị trường hợp tồi nhất. *Mốc tới hạn* của tác vụ τ_i xuất hiện khi nó và mọi tác vụ có độ ưu tiên cao hơn được lập lịch đồng thời. Mô tả mốc tới hạn của τ_5 trong hình 5.13 (vùng rời nét là thực hiện tác vụ, khối biên liền nét là thời gian chờ tác vụ). Nếu tác vụ τ_i phù hợp điểm chết khi được lập lịch theo mốc tới hạn thì nó luôn phù hợp điểm chết trong cách lập lịch khác.

Lý do được minh họa trong hình 5.13. Hãy chú ý điều xảy ra khi lập lịch τ_i nếu di chuyển thời gian đòi hỏi của tác vụ ưu tiên cao hơn τ_h lên hoặc xuống. Nếu cho thời điểm đòi hỏi của τ_h lên t+ε thì tương ứng phải giảm thời gian thực hiện các tác vụ này trong khoảng (t, t+Dτ_i) đi ε. Tuy nhiên tổng số thời gian dành cho thực hiện τ_h trong (t,t+Dτ_i) không tăng lên. Tương tự với việc giảm. Do vậy, tại mốc tới hạn, tổng thời gian các tác vụ có độ ưu tiên cao thực hiện là lớn nhất.

Vì thế, có thể xác định nếu kết quả phân công ưu tiên trong một lịch khả thi bằng cách mô phỏng thực hiện các tác vụ tại mốc tới hạn của tác vụ có độ ưu tiên nhỏ nhất. Nếu tác vụ có độ ưu tiên nhỏ nhất phù hợp điểm chết khi bắt đầu từ khoảng tới hạn thì các tác vụ khác cũng phù hợp điểm chết của chúng. Tác vụ có khoảng thời gian quá ngắn sẽ có ít thời gian thực hiện công việc của nó hơn là tác vụ có khoảng thời gian dài. Bằng trực giác, thấy sẽ tốt hơn nếu gán mức ưu tiên cao cho tác vụ có khoảng thời gian ngắn và ngược lại.

Đặt PR_h là độ ưu tiên của tác vụ τ_h. Giả sử rằng nếu $PR_h>PR_1$, CPU sẽ xử lý τ_h trước τ_l (tức là giá trị PR cao hơn tương ứng mức ưu tiên cao hơn). Qui tắc phân công ưu tiên tác vụ được gọi là phân công độ ưu tiên tốc độ đều RM (Rate Monotonic) như sau:

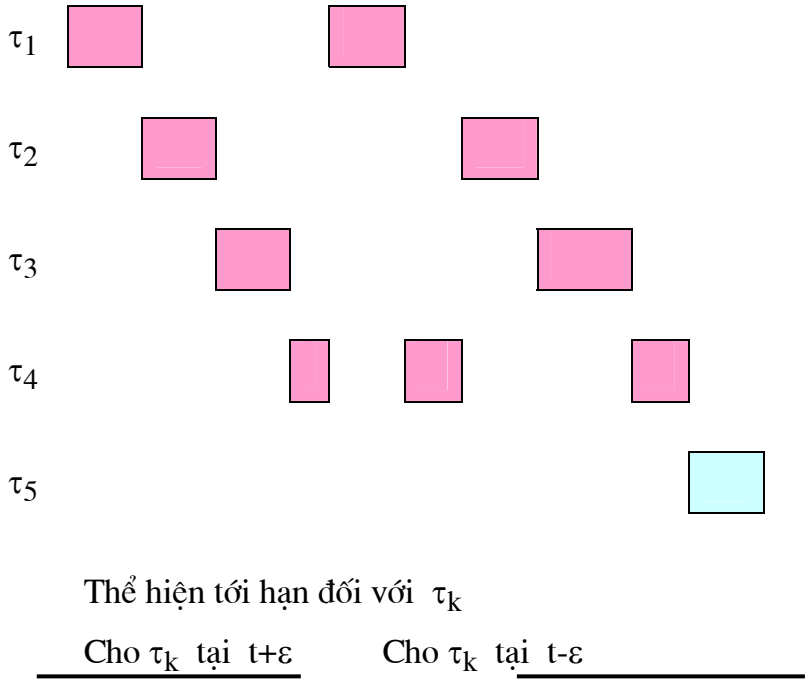
Hệ phân công ưu tiên RM

Nếu $T_h<T_1$ thì $PR_h>PR_1$

Phép phân công ưu tiên tốc dộ đều rất dễ thi hành vì chỉ cần xếp các tác vụ theo độ dài các khoảng thời gian của nó. RM cũng tạo ra một phân công độ ưu tiên tốt. Có thể chỉ ra rằng nếu tồn tại một lập lịch ưu tiên cố định cho một tập các tác vụ có chu kì thì RM sẽ tạo ra một lịch như vậy.

Nói ngắn gọn, RM là một thuật toán phân công độ ưu tiên tối ưu theo nghĩa nếu tồn tại một phân công ưu tiên tĩnh mà lập lịch kết quả là khả thi thì phân công đơn điệu đều cũng sẽ sinh ra lịch khả thi.

Để hiểu được tính chất này của phân công RM, giả sử rằng có một phân công A không RM sản sinh ra một lịch khả thi. Có thể hiển thị các tác vụ đã xếp theo ưu tiên giảm dần mà τ_l cao nhất còn τ_n thấp nhất. Vì A là hệ không-RM nên tồn tại cặp 2 tác vụ τ_i và τ_{i+1} mà $T_i>T_{i+1}$. Nếu thay đổi độ ưu tiên của 2 tác vụ này thì hệ vẫn sẽ là hệ khả thi?



Hình 5.13. Mốc tới han môt tác vu

Nhớ lại, chỉ cần xem nếu τ_i và τ_{i+1} phù hợp điểm chết tại *mốc tới hạn*. Chú ý điều gì xảy ra tại mốc tới hạn theo phân công độ ưu tiên A (hình 5.14). Trong khoảng thời gian $(0,T_{i+1})$ sau mốc tới hạn, tác vụ τ_i sau τ_{i+1} là H giây. Do τ_{i+1} phù hợp điểm chết của nó nên

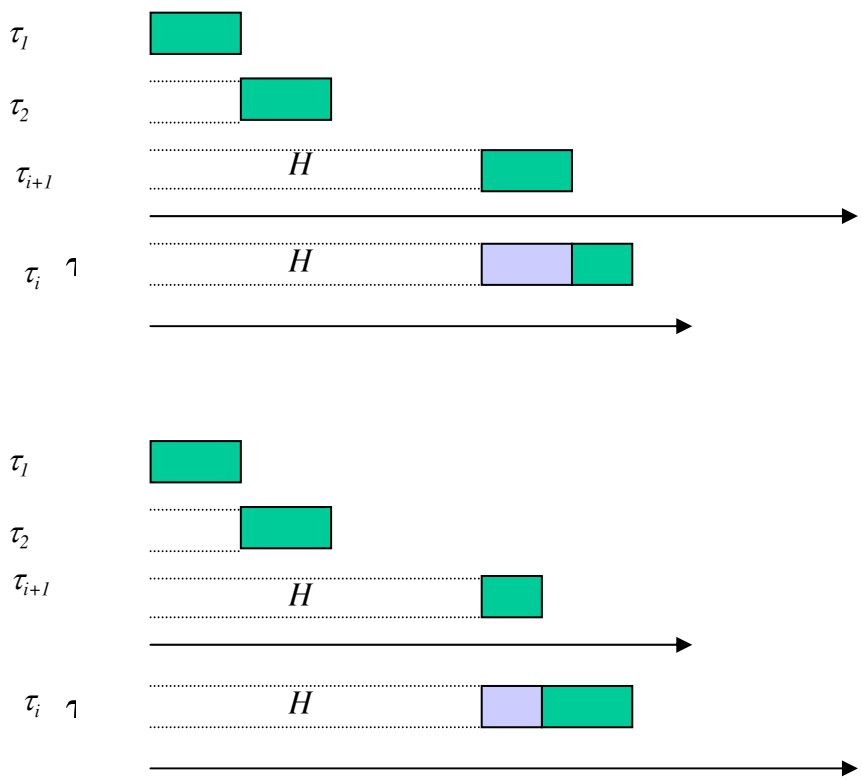
$$H+C_i+C_i+1 \le T_{i+1}$$

Tiếp đó, khi thay đổi độ ưu tiên của τ_i và τ_{i+1}. Tác vụ τ_{i+1} chắc chắn phù hợp điểm chết vì đã tăng độ ưu tiên của nó. τ_i cũng phù hợp điểm chết vì nó sẽ hoàn thành công việc trong khoảng $T_{i+1}<T_i$. Do vậy, việc thay đổi trên vẫn dẫn đến một kết quả lịch khả thi.

Nếu lặp các bước hoán đổi sẽ tạo ra một phân công RM. Suy ra nếu một tập các tác vụ có chu kì có phân công ưu tiên cố định khả thi thì RM sẽ là một trong số các lịch đó.

Phân tích thời gian

Để chắc chắn tất cả các tác vụ phù hợp điểm chết, CPU sẽ được đặt ở chế độ rỗi trong một số chu kì thời gian (những chu kì này được sử dụng thực hiện các tác vụ không tới hạn) Chúng ta cũng phải cần chắc chắn về mức độ tốt của CPU mà thuật toán đem lại (ví dụ thời gian nhàn rỗi không quá dài).



Hình 5.14. Ví dụ về swapping tác vụ

Định nghĩa *tải L* của các tác vụ thời gian thực là phân số thời gian mà các tác vụ này sử dụng CPU

$$L = \sum_{i=1}^{n} \frac{C_i}{T_i}$$

Các nghiên cứu chỉ ra rằng điều kiện đủ để RM là một phân công ưu tiên khả thi là $L \le n(2^{1/n} - 1)$. Có nghĩa là nếu việc tải các tác vụ thời gian thực là tối thiểu thì RM sẽ là một phân công ưu tiên khả thi của chúng ta. Chứng minh điều này khá phức tạp và không cần thiết nên không đề cập đến. Tuy nhiên, biết rằng $n.((2^{1/n} - 1)$ sẽ lớn hơn 69 phần trăm khi n đủ lớn và do vậy RM sẽ không lãng phí quá 31 phần trăm của CPU.

Cận $n.((2^{1/n} - 1)$ tải không tỏ ra tốt, đây chỉ dùng để đánh giá, không phải là điều kiện tiên quyết. Trong ví dụ tập tác vụ V={(1,2), (1,3), (1,6)} có lịch phân công theo RM khả thi và 100 phần trăm CPU dành cho việc tải.

Có thể tìm thấy điều kiện cần thiết để xác định liệu tập tác vụ đã cho có thể lập lịch khả thi được hay không bằng cách thử đặt chúng (tức các tác vụ) vào các mốc tới hạn của chúng. Sắp các tác vụ theo độ ưu tiên giảm dần vào một danh sách. Đặc r_i là thời gian trả lời của tác vụ τ_i tại mốc tới hạn. Nên, τ_i sẽ thực hiện xong sau r_i sau mốc tới hạn. Trong thời gian đó, tác vụ có độ ưu tiên cao τ_h sẽ yêu cầu dịch vụ $[r_i/T_h]$ lần và sẽ cần C_h giây của CPU cho các yêu cầu đó. Suy ra r_i phải thoả mãn mệnh đề sau.

$$r_i = C_i + \sum \left[\frac{r_i}{T_h} \right] C_h$$

Vì r_i xuất hiện ở hai vế của mệnh đề nên tính trực tiếp nó khá là khó khăn. Nhưng có thể tính được r_i theo cách gián tiếp. Đầu tiên, giả sử τ_i không chờ bất kì một tác vụ có độ ưu tiên cao nào. Đặt giá trị r_i vào vế phải mệnh đề và tính được giá trị r_i mới biểu diễn số các dịch vụ đã thực hiện trong C_i đầu tiên sau điểm tới hạn. Trong QT soát lại

thời gian trả lời việc thực hiện một số tác vụ có độ ưu tiên cao có thể xảy ra. Tiếp tục đặt r_i vào vế phải mệnh đề và tiếp tục tính giá trị r_i đến khi ta tìm thấy giá trị ổn định. Các bước trên được biểu diễn tương đương bằng toán học như sau

$r_i(0)=C_i$

$$r_i(k+1) = C_i + \sum_{h=i}^{i-1} \left[\frac{r_i(k)}{T_h} \right] C_h$$

thực hiện đến khi $r_i(k) = r_i(k+1)$. Nếu tại một số bước $r_i(k)>T_i$ thì tập tác vụ không thể được sắp lịch khả thi.

5.5.2. Hệ thống điểm tới han đều

Một số tác vụ trong hệ thống thời gian thực cần hoàn thành công việc thực hiện trước một khoảng thời gian ngắn khi được yêu cầu. Chúng ta có thể tạo ra hệ thống kiểu này bằng cách thay đổi hệ thống các tác vụ có chu kì. Cho

$V\{J_i=(C_i, T_i, D_i)/ 1 \le i \le n\}$

là mô tả cho các công việc thực hiện trong hệ thống. Như phần trước, gán cho các tác vụ thực hiện chương trình i là τ_i. Tác vụ τ_i cần thực hiện một lần trong khoảng thời gian Ti và yêu cầu Ci thời gian của CPU để xử lý. Nếu τ_i được yêu cầu tại thời điểm t, nó phải hoàn thành công việc trước thời gian $t+D_i$ hoặc nó sẽ vượt qua điểm tới hạn.

Cách phân phối tối ưu dựa vào sự cố định độ ưu tiên cho mô hình này được thể hiện bằng thuật toán DM (Deadline Monotonic)

Phân công ưu tiên điểm tới hạn đều

Nếu $D_h<D_1$ thì $PR_h>PR_1$

Trường hợp để DM đạt được tối ưu tương đương như trong trường hợp của RM. Tuy vậy ta sẽ gặp một chút khó khăn khi phân tích DM nhiều hơn so việc phân tích RM bởi vì không có một công thức nào đặt cho công việc này dựa trên sự tải lịch tối ưu tin cậy. May mắn là ta có thể sử dụng *thời gian trả lời* response time (như trong mệnh đề 5.1) mà không cần sự thay đổi nào. Tác vụ gọi là khả thi nếu $r_i \le D_i$ với mọi $i = 1 \to n$.

5.5.3. Ưu tiên điểm tới han sớm nhất

Nếu quyết tâm xây dựng một phương pháp lập lịch phức tạp, phải cần đến sự phục vụ của CPU nhiều hơn so với nhiều phương pháp lập trước đó. Ý tưởng chung là sử dụng phương pháp lập lịch có độ ưu tiên động (thay đổi). Có nghĩa là các quan hệ của độ ưu tiên của các tác vụ có thể thay đổi khi hệ thống hoạt động. Độ ưu tiên có thể đạt giá trị mới khi có sự kiện quan trọng xảy ra, kiểu như tác vụ kết thúc, tác vụ đồng bộ hay đơn giản tại thời điểm bắt đầu tác vụ.

Đặt $\tau_k(i)$ là tác vụ xảy ra trong khoảng thứ i của công việc J_k và đặt $d_k(i)$ là điểm chết của nó. Tương tự, đặt $PR_k(i)$ là độ ưu tiên phân công cho $\tau_k(i)$. Thuật toán phân công động tối ưu gọi là ưu tiên điểm tới hạn sớm nhất (Earliest Deadline First - EDF).

Phân công ưu tiên điểm tới hạn sớm nhất

Nếu $d_h(i)<d_1(j)$ thì $PR_h(i)>PR_1(j)$

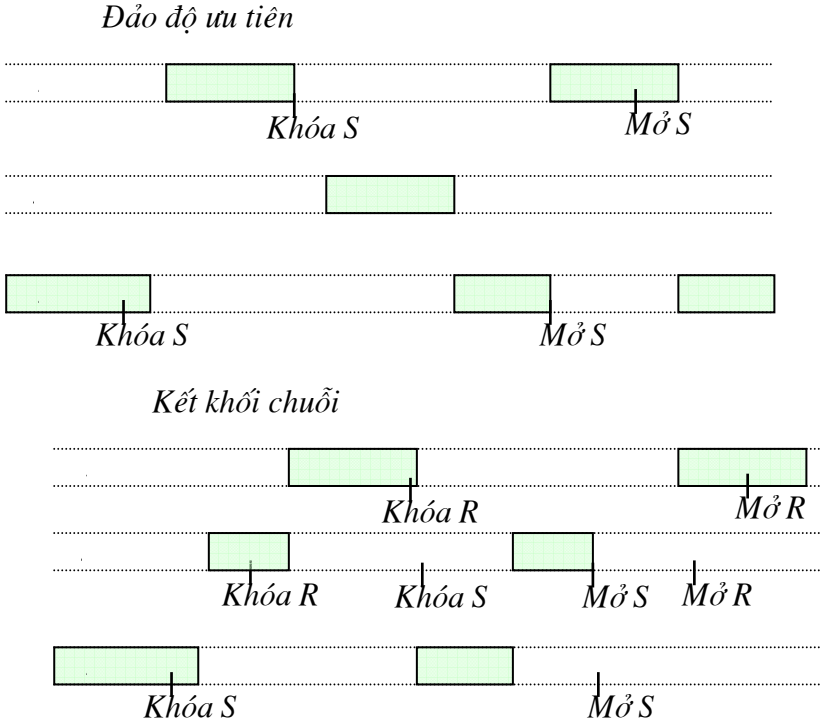
Ta có thể chỉ ra rằng EDF đạt tối ưu như cách ta đã tiến hành với RM và DM. Giả sử rằng tồn tại một cách sắp lịch khả thi không sử dụng phương pháp theo EDF. Khi đó ta sẽ có cặp tác vụ $\tau_k(i)$ và $\tau_l(j)$ mà $d_h(i)<d_1(j)$ nhưng ưu tiên của $\tau_k(i)$ và $\tau_l(j)$. Lặp lại đế khi lịch là EDF.

Thực ra với $D_i=T_i$ cho 1 tập các tác vụ có chu kì thì chúng là EDF (tồn tại cách phân công khả thi theo EDF) miễn là $L \leq 1$.

Để ý EDF sử dụng điểm của các khoảng xảy ra chốc lát của các tác vụ để xác định các độ ưu tiên, không phải là thông tin về chính các khoảng đó. Do vậy, EDF thích hợp cho lập lịch có chu kì. Theo chứng minh EDF tối ưu được khi chỉ sử dụng điểm chết của các khoảng xảy ra chốc lát của các tác vụ. EDF là thuật toán tốt cho lập lịch các tác vụ thời gian thực có chu kì.

5.5.4. Đồng bộ thời gian thực

Như đã biết ở các mô hình tác vụ trước, ta đã giả sử mỗi tác vụ là độc lập với nhau. Tuy nhiên, một tập các tác vụ cũng có thể liên kết thực hiện một mục đích nên cần chia xẻ thông tin và do vậy phải thiết lập một cơ chế đồng bộ. Hơn nữa mỗi tác vụ cũng cần chiếm hữu các tài nguyên hệ thống (bộ nhớ, kênh truyền,…) trong QT hoạt động. Đồng bộ giữa các tác vụ thời gian được hiểu là phát hoặc thu semaphore (cờ tín hiệu). Không may, việc làm này có thể dẫn đến sự chậm trễ thời gian tuy rất ngắn trong QT đồng bộ.



Hình 5.15. Kế khối chuỗi và đảo đô ưu tiên

Xem xét hệ thống bao gồm t_1, t_2 và t_3 đã sắp theo thứ tự giảm của độ ưu tiên và semaphore S. Giả sử t_3 đã khoá S lại kh t_1 được yêu cầu. Vì τ_1 có độ ưu tiên cao nhất nên nó sẽ được thực hiện. Nếu τ_1 thử khoá S, nó sẽ phải dừng lại và chờ cho đến khi τ_3 tháo khoá S. Tuy nhiên, trong trường hợp τ_3 liên quan đến τ_2 (ví như đã sẵn sàng trên hàng đợi) thì nó sẽ phải dừng việc tháo khoá S cho đến khi τ_2 hoàn thành. Suy ra τ_2 hoạt động có sự liên hệ phụ thuộc với τ_1. Điều này gọi là sự đảo ngược ưu tiên bởi vì τ_2 đã có độ ưu tiên thực hiện cao hơn τ_1 trong một khoảng thời gian. Một cách khác có thể gọi đây là *chuỗi khống chế*. Coi τ_1 khoá semaphore R, τ_2 khoá R và S, τ_3 khoá S thì τ_1 sẽ coi như là bị khoá bởi τ_3 dù rằng τ_1 không khoá S.

Giao thức đồng bộ thời gian thực tương tác với lịch nhằm giảm bớt độ đảo ngược ưu tiên về mức nhỏ nhất và có thể suy đoán được. Việc tương tác này hoàn thành bằng cách điều chỉnh độ ưu tiên của các tác vụ thời gian thực và cung cấp một cách chọn lựa các lối vào sử dụng semaphore.

Tác vụ τ_i sẽ truy cập tới một tập các phiên tới hạn c_i,$\{z_i(k)/\ 1 \leq k \leq c_i$. Các phiên tới hạn có khả năng chồng chéo lên nhau trong cùng một vụ cho phép sự xâm nhập vào các tài

nguyên khác nhau trong cùng lúc. Tuy nhiên, việc chồng chéo phiên tới hạn phải dựa theo cơ chế đóng tổ tức là nếu $z_i(1)$ và $z_i(2)$ chồng chéo và $z_i(1)$ khởi đầu trước $z_i(2)$ thì $z_i(2)$ phải kết thúc trước $z_i(1)$.

Tác vụ τ_h sẽ truy cập tới một tập các phiên tới hạn $z_i(2)$ của tác vụ có độ ưu tiên nhỏ hơn τ_1 nếu τ_h phải đợi τ_1 để thoát ra $z_1(k)$ trước khi tiếp tục thực hiện. Tác vụ τ_h có thể bị khống chế bởi $z_1(k)$ bằng cơ chế không trực tiếp như kiểu chuỗi chế hoặc bằng mô phỏng giao thức đồng bộ thời gian thực. Nếu $z_1(k)$ được bảo vệ bằng semaphore S thì τ_h bị khống chế thông qua S.

Giao thức kế thừa ưu tiên

Trong ví dụ trước, sự khống chế tác vụ τ_1 xảy ra không dự đoán được trong một thời gian dài bởi vì τ_2 có ưu thế hơn τ_3 trong khi τ_3 lại khống chế τ_1. Tức là, τ_3 có quyền ưu tiên hơn τ_2 trong khi τ_3 khống chế τ_1. Do vậy, τ_3 thừa kế độ ưu tiên của τ_1. Từ đó dẫn đến *giao thức kế thừa ưu tiên* PIP (Priority Inheritance Protocol) có các luật sau:

- Một tác vụ được phân công thường (RM) khi nó được yêu cầu.
- Phân công cho CPU độ ưu tiên cao nhất

Trước khi một tác vụ bước vào phiên tới hạn, nó đầu tiên phải yêu cầu khoá semaphore các phiên tới hạn khác

Nếu tác vụ τ_h bị khống chế thông qua S được điều khiển bởi τ_1 thì nó sẽ bị chuyển ra khỏi danh sách sàng và PR_1 sẽ được phân công thay cho PR_h (thừa kế độ ưu tiên của τ_h)

- Kế thừa độ ưu tiên có thể mở rộng.

Tác vụ có độ ưu tiên cao nhất bị khống chế thông qua S sẽ được cho vào hàng sẵn sàng. Tác vụ τ_1 trả lại độ ưu tiên mà nó kế thừa qua S và tiếp tục ở độ ưu tiên nhỏ hơn.

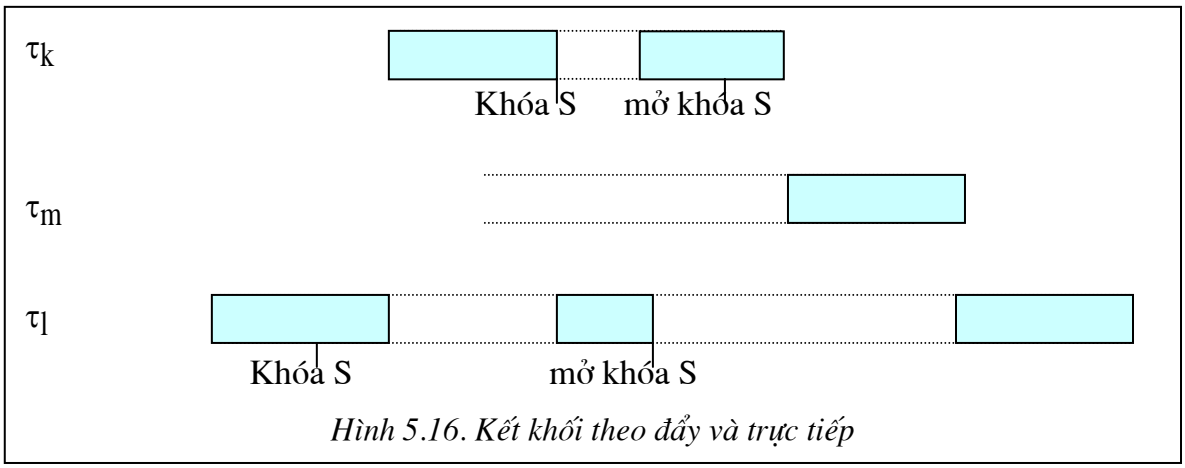
Giao thức thừa kế độ ưu tiên giới hạn thời gian trong QT tác vụ bị khống chế (vd: thời gian lúc tác vụ có độ ưu tiên thấp đang hoạt động). Nếu tính được quãng thời gian lớn nhất mà τ_i có thể bị khống chế, ta có thể có sự tính toán trước đó để khẳng định có tồn tại một cách sắp lịch khả thi cho tập tác vụ này không? Có hai phương pháp chỉ ra một tác vụ có độ ưu tiên thấp khống chế tác vụ có độ ưu tiên cao hơn. PHương pháp đầu tiên là khống chế trực tiếp. Nó xảy ra khi tác vụ có độ ưu tiên cao muốn khoá semaphore đã được quản lý bởi tác vụ có độ ưu tiên thấp. Phương pháp tứ hai liên quan đến giao thức thừa kế độ ưu tiên. Khống chế *đẩy qua xảy ra* khi một tác vụ có độ ưu tiên thấp thừa kế được mức ưu tiên cao và nó là tác vụ có độ ưu tiên trung bình có khống chế đẩy qua. Chú ý rằng khi tác vụ có độ ưu tiên thấp nhả semaphore, tác vụ có độ ưu tiên cao thực hiện và tác vụ có độ ưu tiên thấp bởi vì các tác vụ này hoạt động theo sự sắp xếp độ ưu tiên. (vd trong hình 5.16 khi τ_h yêu cầu khoá S, nó bị khống chế trực tiếp bởi τ_1 qua S và τ_m bị khống chế đẩy qua bởi τ_1 qua S).

Một tác vụ có độ ưu tiên cao τ_h bị khống chế bởi tác vụ τ_1 có độ ưu tiên thấp hơn nếu và chỉ nếu τ_1 đang thực hiện trong phiên tới hạn và τ_h được yêu cầu. Điều này xảy ra vì τ_h bị cấm, τ_1 được thực hiện chỉ khi τ_h bị khống chế bởi nó (vì không thể gia tăng độ ưu tiên cho τ_1)

Ta có thể tạo ra các quan sát mà qua đó tính ra thời gian khống chế tối đa. Tác vụ τ_1 khống chế tác vụ τ_h trong 2 trường hợp. Giả sử τ_1 khoá R và S trước khi τ_h được yêu

cầu. Khi τ_h thực hiện, 2 tác vụ có độưu tiên cao hơn τ_1 và τ_2 sẽ được yêu cầu. Giả sử τ_1 muốn khoá S, τ_1 thừa kế PR_1 cho đến khi nó nhả S. Sau đó τ_2 muốn khoá R nên τ_1 thừa kế PR_2 đến khi nó nhả R. Tuy vậy, khống chế đẩy qua τ_h bởi τ_1 là không quan trọng vì khoảng thời gian tương tự cho việc khống chế không xảy ra nếu chỉ τ_2 được yêu cầu.

Tác vụ τ_1 có thể khống chế τ_h chỉ khi τ_1 đang trong phiên tới hạn. Vì ta đã giả thiết các phiên tới hạn dựa theo cơ chế đóng tổ nên suy ra biết được phiên tới hạn dài nhất mà trong đó τ_1 khống chế τ_h . Đặt $B_h(1)$ là tập tất cả các phiên tới hạn mà τ_1 ở trong đó có thể khống chế τ_h. $B_h(1)$ bằng rỗng nếu $PR_1>PR_h$. Việc khống chế được chia làm hai loại trực tiếp và đẩy qua nên chúng ta phải phân tích tập tất cả các tác vụ khoá semaphore S.



Hình 5.16. Kết khối theo đẩy và trực tiếp

Cho ceiling (S) là độ ưu tiên của tác vụ có độ ưu tiên cao nhất có thể bị khống chế bởi S. Khoảng tới hạn $z_1(j)$ của τ_1 khống chế nếu $z_1(j)$ được bảo vệ bằng S và ceiling (S)$\geq PR_h$. Nếu không tồn tại các phiên theo cơ chế đóng tổ, ceiling (S) là độ ưu tiên của tác vụ có độ ưu tiên cao nhất khoá S. Nếu có một số các phiên theo cơ chế đóng tổ sẽ gây ra chuỗi khống chế (như thấy ở hình 5.16). Trong trường hợp này, cần các phân tích phức tạp hơn.

Đặt $E_h(1)$ là thời gian thực hiện lớn nhất của các khoảng tới hạn thuộc $B_h(1)$. B_h là quảng thời gian dài nhất mà τ_h bị khống chế. Suy ra

$$B_h \leq \sum_{\{lPR_1 < PR_h} E_h(l)$$

Không tồn tại cách tính B_h nào khác có thể đưa ra cận tốt hơn. Thay vì xem xét các khống chế sinh ra bởi các tác vụ, ta có thể xem xét các khống chế gây ra bởi các phiên tới hạn. Giả thiết τ_h bị khống chế thông qua semaphore S. S bị trông giữ bởi τ_1 và khống chế kết thúc khi τ_1 nhả S. Sau khi nhả S, không một tác vụ có độ ưu tiên thấp nào được thực hiện đến khi τ_h thực hiện xong. Do vậy, τ_h có thể bị khống chế qua S nhiều nhất một lần.

Đặt E_h (S) là quãng thời gian thực hiện dài nhất của các khoảng tới hạn bảo vệ bởi S bằng tác vụ có độ ưu tiên tháp Hown PR_n (bằng 0 nếu không có tác vụ nào như vậy). Khi đó

$$B_h \leq \sum_{\{S \setminus ceiling(S) \geq PR_n\}} S_h(S) \quad (5.3)$$

Chúng ta có thể không liên kết thời gian khống chế tối đa B_h vào phương án sắp lịch chỉ với những thay đổi không quan trọng. Khoá tìm kiếm là khi quãng thời gian khống chế tăng lên cho đến khi tác vụ τ_h hoàn thành công việc, CPU không bị lãng phí trong suốt QT. Điều kiện cần thiết cho một phân công RM (như trong phần trước) có thể mở rộng như sau với h=1…n

$$\sum_{j=1}^{h} \frac{C_j}{T_j} + \frac{B_h}{T_h} \le h\left(2^{1/n} - 1\right)$$

Trong đó $PR_h > PR_{h-1}$ với h=2…n

Với những quan sát kĩ lưỡng hơn, ta có thể thay đổi thời gian đáp ứng tính toán ở mệnh đề 5.1 là

$r_i(0)=C_i+b_i$

$$r_i(k+1)=C_i+B_i+\sum_{h=1}^{i-1}\left\lceil \frac{r_i(k)}{j_h} \right\rceil C_h \quad (5.4)$$

Tương tự như với RM và DM, tập các tác vụ là khả thi nếu thời đáp ứng luôn luôn nhỏ hơn hay bằng với điểm chết của tác vụ

5.5.5. Giao thức mức ưu tiên trần

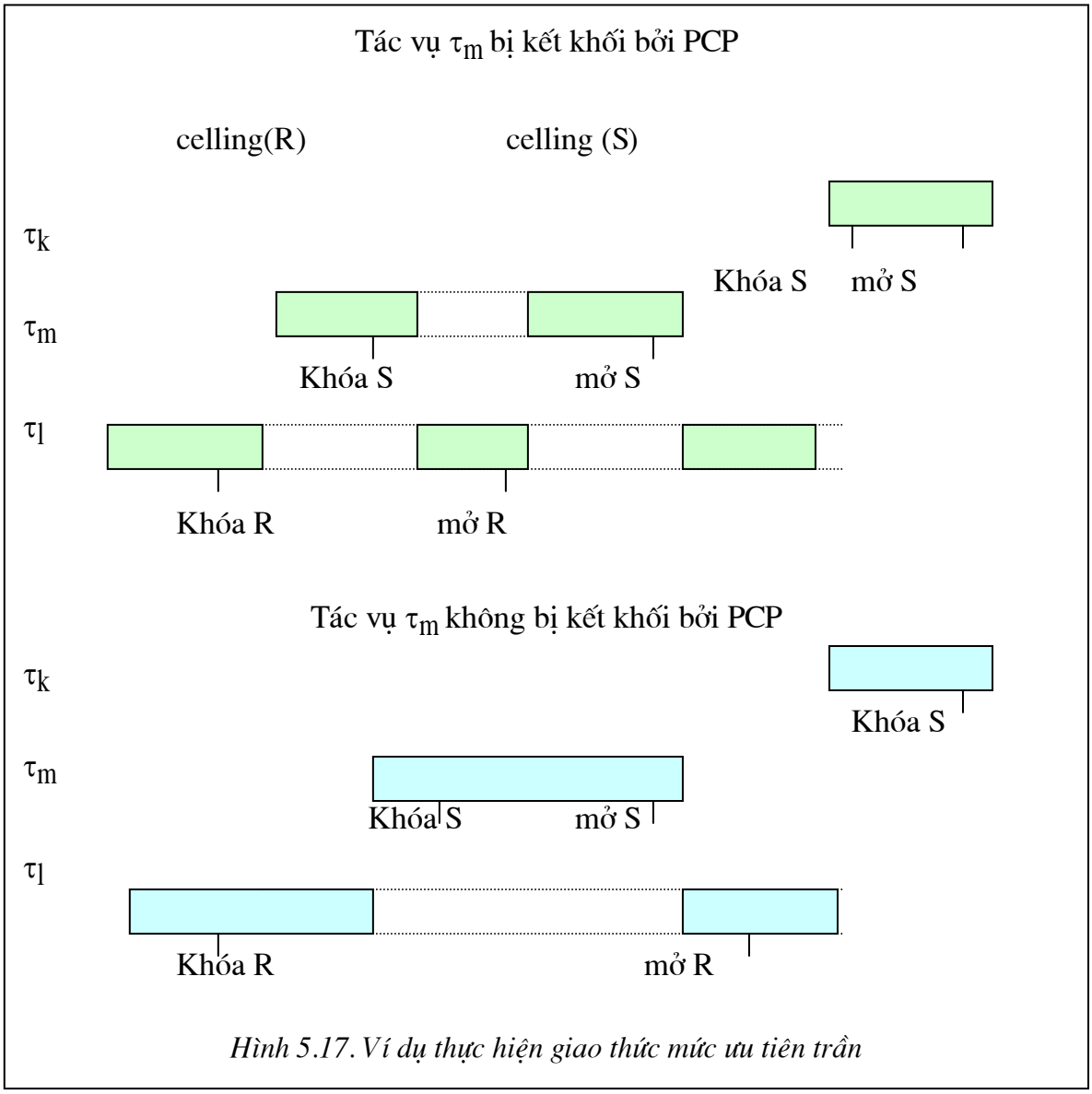
Khi giao thức thừa kế ưu tiên hạn chế các khống chế, khoảng trống B_h vẫn còn khá dài. Giao thức mức ưu tiên trần (PCP) có thể biết sự giới hạn B_h ở trong một phiên tới hạn. Để hoàn thành nó, PCP sẽ thêm các hạn chế hà khắc khi tác vụ muốn đặt vào phiên bản tới hạn. Tuy nhiên, nhưng tới hạn đó cần phải được xem xét một cách tổng thể.

Khi tác vụ được yêu cầu (sử dụng PIP), có thể có một số các tác vụ có mức ưu tiên thấp τ_1 đang khoá semaphore S mà có thể khống chế τ_h. Chúng ta phải khẳng định khi τ_h được yêu cầu, tối đa một tác vụ có độ ưu tiên thấp khoá semaphore S đang khống chế τ_h. Đòi hỏi này đặt ra một luật đơn giản: một tác vụ muốn khoá S chỉ khi không có tác vụ nào khác khoá R mà tác vụ τ_h có thể bị khống chế thông qua S và R.

Luật khống chế trên yêu cầu cần có một phương thức đơn giản qua đó xác định các tác vụ nào bị semaphore khống chế. May mắn là ta đã có một phương thức như thế: ceiling (s) chính là độ ưu tiên của tác vụ có độ ưu tiên cao nhất mà có thể bị khống chế bởi S. Suy ra, τ_h chỉ có thể bị khống chế qua S nếu và chỉ nếu ceiling (S)≥ PR_h. Khi đó ta gọi ceiling(S) là mức ưu tiên tới hạn của S.

Khi τ_1 muốn khoá S, nó sẽ kiểm tra mức ưu tiên trần của tất cả các tác vụ bị khống chế qua semaphore R. Nếu mức ưu tiên trần của R nhỏ hơn PR_1, R không thể khống chế tác vụ mà τ_1 có thể khống chế thông qua S. Do vậy, tác vụ có độ ưu tiên cao τ_h có thể bị khống chế qua S và R.

Một ví dụ minh chứng cho việc sử dụng luật PCP ở hình 5.17. Trong bước thực hiện tại phía trên của hình vẽ, tác vụ τ_m bị khống chế qua semaphore S bởi vì τ_1 giữ R và ceiling(R)≥ PR_m. Nếu τ_m bị cấm khi yêu cầu S, τ_h có thể bị khống chế bởi khoảng thời gian xác định của hai hoạt động phiên tới hạn. Tuy τ_m bị khống chế bằng cách khoá như vậy nhưng nó không bao giờ thoát ra được. PCP bảo đảm các tác vụ chỉ bị khoá nhiều nhất một lần. Trong hoạt động ở phía dưới hình vẽ, ceiling(R)<PR_m nên τ_m có thể giữ S mà không sợ bị khống chế τ_h hai lần.



Hình 5.17. Ví dụ thực hiện giao thức mức ưu tiên trần

Theo quan điểm này, ta đã hâu như thiết lập ra giao thức mức ưu tiên tới hạn ngoài việc thêm vào một số luật kèm theo sau:

- Mỗi semaphore S có một liên kết mức ưu tiên tới hạn ceiling(S) với nó.

- Khi τ_1 muốn khoá S, việc này chỉ đúng khi PR_1 lớn hơn ceiling(R) với mọi semaphore R bị khoá bởi các tác vụ khác. Nếu không τ_1 sẽ bị khống chế.

- Nếu tác vụ τ_h bị khống chế qua S bị giữ bởi τ_1 thì τ_1 sẽ thừa kế độ ưu tiên PR_h.

- Khi τ_1 nhà S, τ_1 nhà tất cả các độ ưu tiên mà nó kế thừa qua S. Tác vụ có độ ưu tiên cao nhất sẽ được đặt vào hàng đợi

Nếu ra tính được quãng thời gian khống tối đa cho mỗi tác vụ, ta sẽ sử dụng mệnh đề 5.4 để xác định rằng liệu tập các tác vụ có một cách sắp lịch khả thi sử dụng phân công ưu tiên hay không? Bởi vì một tác vụ có thể bị khống chế bởi nhiều nhất một tác vụ khác và qua nhiều nhất một semaphore nên ta có

$$B_h \le \max\{E_h(S) \mid ceiling(S) \ge PR_h\} \quad (5.5)$$

CHƯƠNG VI. HỆ THỐNG FILE PHÂN TÁN

Hệ thống tính toán là tập hợp các thao tác xử lý trên đối tượng dữ liệu. Các đối tượng bền vững cần được lưu giữ lâu dài để tìm kiếm. Chúng cần được đặt tên và bảo quản trên các thiết bị lưu trữ ổn định, chẳng hạn như đĩa từ. Các đối tượng dữ liệu có tên được gọi là file. Giả sử file là đối tượng dữ liệu cơ bản, cấu trúc nội tại và thể hiện chúng được coi như vấn đề thi hành của đặc tả hệ thống. Hệ thống file là thành phần chính trong HĐH, có trách nhiệm đặt tên, tạo mới, xoá, tìm kiếm, sửa chữa và bảo vệ mọi file trong hệ thống. File cần được chia xẻ để dễ dàng cộng tác và được phân bố tại các nút rời rạc trong hệ thống phân tán.

Hệ thống file phân tán (DFS) là thi hành của hệ thống file, phù hợp với việc phân tán vật lý trên các nút lưu giữ song cung cấp một cái nhìn của hệ thống file tập trung theo truyền thống cho người dùng. Sự tồn tại các file trong phạm vi của hệ thống là trong suốt đối với người dùng. Nhiều khái niệm quan trọng trong thiết kế hệ phân tán được sáng tỏ nhờ thi hành DFS. Thứ nhất, DFS sử dụng nhiều khía cạnh của tính trong suốt. Thứ hai, dịch vụ thư mục trong DFS là ví dụ tốt của dịch vụ tên, một thành phần cốt yếu trong hệ phân tán. Thứ ba, yêu cầu về hiệu năng và độ sẵn sàng cần đến bộ đệm cache và nhân bản, dẫn tới bài toán quản lý kết dính cache và nhân bản. Thứ tư, điều khiển truy cập và bảo vệ đối với DFS mở ra nhiều bài toán quan trọng trong an toàn hệ phân tán. Chương này giới thiệu những vấn đề này (tại các chương thuộc phần 2 tài liệu, các vấn đề này sẽ được nghiên cứu chi tiết hơn).

6.1 Đặc trưng của DFS

Phân tán và *vô số* cả về người dùng và file là hai đặc trưng quan trọng của DFS. Đa khách định vị phân tán truy nhập file phân tán (thường được nhân bản). Mục tiêu thiết kế là che dấu đi hai đặc trưng phân tán và vô số đối với người dùng. DFS trong suốt có các tính trong suốt sau:

a. Khách phân tán

Người dùng đăng nhập tại máy tính bất kỳ (người dùng có quyền sử dụng máy tính đó) trong hệ thống với một thủ tục đăng nhập đồng nhất và nhận được cái nhìn đồng nhất về hệ thống file mà không phụ thuộc vào máy tính đăng nhập. Đặc tính này gọi là *trong suốt đăng nhập*. xxxMỗi khi tại máy chủ, QT khách chạy trên máy cục bộ có cơ chế đồng nhất để truy cập tất cả các file trong hệ thống, bất kể file ở cục bộ hay ở xa. Tính chất này gọi là *trong suốt truy nhập*.

b. File phân tán

Tên được đặt cho file không chứa thông tin về vị trí vật lý của file, file là *trong suốt định vị* đối với khách. Hơn nữa, file có thể di chuyển từ vị trí vật lý này sang vị trí vật lý khác mà không phải đổi tên. Tính chất này được gọi là *độc lập định vị* và là tính chất mạnh hơn so với trong suốt định vị.

c. Vô số người dùng

Nhiều người dùng có thể đồng thời sử dụng một file. Một cập nhật file của QT này không đối nghịch việc thực hiện đúng đắn của các QT khác đang đồng thời chia xẻ file này. Tính chất này được gọi là *trong suốt đồng thời*. Những ứng dụng-giao dịch đòi hỏi rằng ứng dụng trình diễn truy nhập file một cách cô lập (bất chấp sự truy nhập file chen ngang của các ứng dụng khác). Bài toán tin cậy thực hiện đồng thời của một giao dịch được chỉ dẫn như *điều khiển đồng thời* trong hệ CSDL. Đây là yêu cầu tới DFS nhằm hỗ trợ *trong suốt đồng thời* tại mức giao dịch.

d. Vô số file

File trong DFS được nhân bản để cung cấp độ dư thừa đảm bảo tính sẵn sàng và cho phép truy nhập đồng thời hiệu quả. DFS với *trong suốt nhân bản* thực hiện cập nhật nguyên tử trên các bản sao và khách không nhận thấy sự tồn tại của các bản sao.

Dãy tính trong suốt như được giới thiệu là đòi hỏi quan trọng khi thiết kế DFS. Tính chất mong muốn khác về đặc trưng trong suốt là thứ lỗi, phân cấp và hỗn tạp của hệ thống. Lỗi, chẳng hạn đổ vỡ các QT phục vụ hoặc khách, mất thông điệp, việc chia cắt mạng, là không ảnh hưởng tới người dùng ngoại trừ việc làm giảm không đáng kể hiệu năng hệ thống. Việc tăng trưởng file và lượng cập nhật không làm ngắt các thao tác thông thường đối với hệ thống file. Lưu ý là vấn đề trong suốt không thể chỉ xét trong hệ thống file phân tán mà cần được xem xét trong hệ thống phân tán nói chung.

6.2 Thiết kế và thi hành DFS

Trong đoạn này, đầu tiên đề cập một số khái niệm cơ bản về file và hệ thống file. Vấn đề duy nhất trong thiết kế và thi hành DFS là dựa theo nhu cầu chia xẻ và nhân bản file. Trọng tâm chính của chương là các giao thức đạt được tính trong suốt trong chia xẻ và nhân bản file.

6.2.1 File và hệ thống file

Với người dùng, file gồm 3 thành phần logic

Tên file	Thuộc tính file	Dữ liệu

File được tạo ra gắn với tên tượng trưng (tên). Khi truy cập file, tên file được ánh xạ tới số hiệu file duy nhất (ufid hoặc thẻ file), cho phép định vị được file vật lý. Đây là chức năng nguyên thủy của *dịch vụ thư mục* trong hệ thống file. Các thuộc tính file điển hình là thông tin về chủ nhân, kiểu, kích thước, tem thời gian và quyền truy nhập file.

Các đơn vị dữ liệu trong file được tổ chức theo cấu trúc phẳng dòng *byte* hoặc dãy tuần tự các *khối*, hoặc theo một lựa chọn khác, là *cấu trúc phân cấp* các bản ghi có chỉ số. Phụ thuộc vào cấu trúc file hạ tầng mà file được truy cập theo một trong ba cách:

(1) *Truy nhập tuần tự*: Trong truy nhập tuần tự, với mỗi file đã mở, hệ thống duy trì một con trỏ định vị file nhằm chỉ dẫn vị trí của đơn vị dữ liệu tiếp theo sẽ được truy cập. Con trỏ file không phải một phần của thuộc tính file; thực chất nó là phần của trạng thái QT (dù cho con trỏ file được chia xẻ giữa các QT liên quan). Mở file khởi tạo con trỏ file và bắt đầu phiên làm việc, truy nhập file tiếp theo được diễn ra cho đến khi file được đóng. Phiên làm việc tương tự kết nối trong truyền thông mạng định hướng kết nối.

(2) *Truy nhập trực tiếp*: Khác với truy cập tuần tự (vị trí đơn vị dữ liệu đọc/ghi đã xác định), trong truy nhập trực tiếp cần chỉ dẫn rõ ràng đơn vị dữ liệu kích thước-cố định bằng số hiệu khối của chúng. Mỗi thao tác đọc/ghi chứa thông tin địa chỉ đầy đủ và độc lập với thao tác đọc/ghi khác. Truy nhập file giống như truyền thông mạng không kết nối. Mở file là không nghiêm ngặt ngoại trừ phiên (xem trong 6.2.5) cần có ngữ nghĩa mong muốn để chia xẻ file. Bổ sung tới định nghĩa phiên, hầu hết các hệ thống file dùng lời gọi hệ thống *open* để thực hiện ánh xạ một lần tên file tới trình bày máy nội tại cho truy nhập file về sau và thu được quyền truy nhập file. Đối với phương pháp truy nhập trực tiếp, thao tác *open* có chức năng giống định vị file *look-up* hơn.

(3) *Truy nhập tuần tự số hiệu (chỉ số kế tiếp)*: Các đơn vị dữ liệu được địa chỉ trực tiếp bằng số hiệu (khoá) gắn với mỗi khối dữ liệu. Truy nhập tuần tự số hiệu đòi hỏi duy trì chỉ số tìm kiếm trong file, cần tìm được định vị khối cho mỗi truy nhập. Tuỳ thuộc vào dung lượng và tổng phí thời gian, truy nhập chỉ số thường được dùng đối với hệ thống

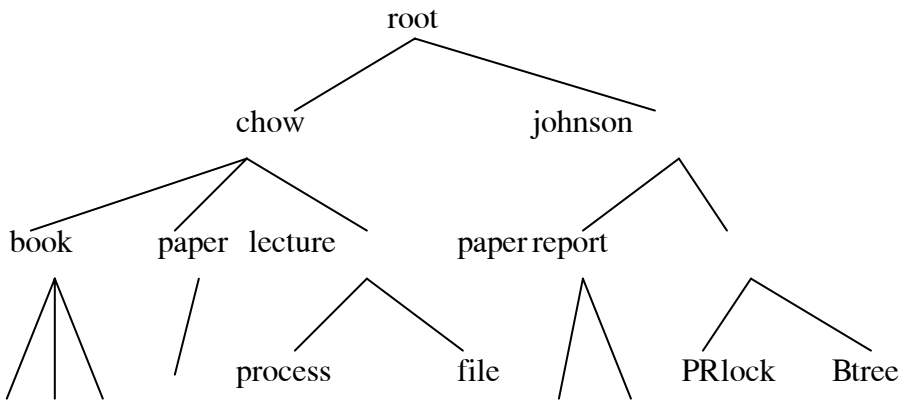
file lớn trong hệ thống máy tính lớn. Để giảm kích thước số hiệu tìm kiếm, thông thường dùng sơ đồ hai mức, là phương pháp truy nhập chỉ số kế tiếp. Một dãy cặp khóa/đối tượng được bảo quản trong khối dữ liệu lớn. Cặp khóa/đối tượng được định vị khi dùng số hiệu tìm kiếm để định vị khối chứa cặp này, sau đó truy nhập dữ liệu trên khối cho đến khi tìm thấy cặp khóa/đối tượng. Thao tác hai mức này tổ hợp các phương pháp truy nhập trực tiếp và tuần tự tương tự như tìm thông tin theo số hiệu trong một quyển sách. Cách thức này được dùng đối với file dữ liệu lớn, trong đó các bản ghi của file được giữ trong bộ nhớ phụ còn bảng số hiệu được duy trì trong bộ nhớ nguyên thủy để tìm kiếm hiệu quả.

Cấu trúc file dãy có lợi ích và thông dụng hơn do tính đơn giản và tương đồng với các thiết bị lưu giữ thi hành file. Ví dụ, Unix coi rằng file là dòng kí tự của chương trình ứng dụng và dãy các khối kích thước cố định trong hệ thống file. Cả hai phương pháp truy nhập tuần tự và trực tiếp được hỗ trợ. Các phương pháp truy nhập khác được xây dựng dựa trên nền cấu trúc file dãy.

Dịch vụ thư mục		Giải pháp tên, thêm và xoá file
Dịch vụ xác thực		năng lực và/hoặc danh sách điều khiển truy nhập
Dịch vụ file	giao dịch	Quản lý đồng thời và nhân bản
	cơ sở	đọc/ghi file và nhận/đặt thuộc tính
Dịch vụ hệ thống		Quản lí thiết bị, cache, khối

Hình 6.1 Thành phần chính trong hệ thống file

Hệ thống file tổ chức và cung cấp dịch vụ truy nhập và bảo vệ cho tập hợp file. Bốn thành phần dịch vụ chức năng chính: thư mục, xác thực, file và hệ thống. Hệ thống file hỗ trợ giao dịch lại phân chia dịch vụ file thành dịch vụ giao dịch và dịch vụ file cơ sở. Dịch vụ giao dịch có bài toán thi hành riêng (xem 6.3.3). Hình 6.1 cho ví dụ về chức năng được mỗi dịch vụ cung cấp. Tổ chức file dữ liệu trong hệ thống file có thể là dãy hoặc phân cấp, tương tự như lựa chọn cấu trúc nội tại của mỗi file riêng. Tuy nhiên, tự nhiên hơn khi tổ chức file theo nhiều mức, nhận được kết quả là thư mục phân cấp và cấu trúc tên. File được đặt tên và truy nhập theo tên đường dẫn phân cấp chẳng hạn /chow/lecture/file, như hình 6.2.



Hình 6.2. Cấu trúc phân cấp hệ thống File

Để truy nhập file, đầu tiên dùng *dịch vụ thư mục* để định vị file. Dịch vụ thư mục ánh xạ tên phân cấp tới địa chỉ là hoàn toàn độc lập với thao tác file thực sự. Do đó tách dịch vụ thư mục từ dịch vụ file nhằm đạt được tính đơn thể và khả chuyển. Ưu điểm chính của việc tách các dịch vụ là cho phép một dịch vụ file chung hỗ trợ được nhiều

kiểu cấu trúc thư mục khác nhau. Thư mục là "file" chứa tên và địa chỉ của các file và thư mục con. Thao tác file thư mục là tra cứu, thêm, xóa các thực thể thư mục (điểm vào) trỏ tới các file. Phục vụ thư mục là khách của dịch vụ file để cập nhật file.

Cập nhật file phải an toàn - đó là vai trò của dịch vụ xác thực. Vị trí logic của dịch vụ xác thực cần ở giữa dịch vụ thư mục và dịch vụ file để bảo vệ file hay ở trước dịch vụ thư mục để bảo vệ thư mục. Trong thi hành thực sự (1) Dịch vụ xác thực được trộn với dịch vụ file: dịch vụ file duy trì thông tin điều khiển truy nhập. Dịch vụ thư mục đơn thuần thực hiện giải pháp tên và cung cấp thẻ file để định vị file được đòi hỏi trong một phục vụ file. Thao tác file được kiểm tra để xác thực khách đưa ra thao tác đó; (2) Dịch vụ xác thực được trộn với dịch vụ thư mục: Dịch vụ thư mục được bổ sung giải pháp tên để thực hiện việc kiểm tra điều khiển truy nhập. Dịch vụ thư mục duy trì thông tin điều khiển truy nhập file (và thư mục). Nếu xác thực khách để mở file đã được kiểm tra, khách nhận được một thẻ file và thẻ đặc quyền không làm giả để sử dụng file của khách. Đối với các truy nhập tiếp theo của dịch vụ file thì chỉ có thẻ đặc quyền là có giá trị đối với phục vụ file. Định danh khách là không thích hợp tới phục vụ file, do thẻ đặc quyền đã xác thực khách. Quy tắc xác thực truy nhập được thảo luận chi tiết trong phần an toàn hệ điều hành.

Dịch vụ file cần cung cấp các thao tác file cơ bản là read/write khối dữ liệu và get/set thuộc tính file. Do file cần được khởi tạo trước khi được dùng và cần được xoá khi không cần thiết, dịch vụ file cũng cần hỗ trợ thao tác tạo và xóa file. Tạo/xóa file bao gồm cả việc bổ sung và xóa bỏ thực thể trong thư mục nhờ dịch vụ thư mục. Chúng liên quan với dịch vụ hệ thống bên dưới là định vị và gỡ bỏ (giải định vị) bufer và file. Dịch vụ file trở thành khách của dịch vụ thư mục và dịch vụ hệ thống.

Thao tác mở file là một thao tác đáng quan tâm trong dịch vụ file. Thao tác mở file bao gồm khởi tạo phiên làm việc các thao tác file lên một file. Nó tương tự việc thiết lập kết nối khách tới phục vụ file. Dịch vụ thư mục được thao tác mở file tra cứu chỉ một lần duy nhất. Thẻ file nhận được khi tra cứu dịch vụ thư mục và giữ tại nhân của khách. Các thao tác read/write tiếp theo được gửi tới dịch vụ file trực tiếp khi dùng thẻ file. Nhân lưu đường đi của kết nối, bao gồm cả con trỏ định vị file cho thao tác read/write tiếp theo. Nhiều hệ thống file còn ngầm định rằng thao tác mở file sẽ khởi tạo file mới nếu nó chưa tồn tại. Thao tác đóng file kết thúc một phiên mở. Thao tác mở file là không thực sự cần thiết nếu quan hệ khách và phục vụ là không kết nối. Trong trường hợp như thế, read và write là các thao tác thực hiện ("ăn") ngay. Mỗi yêu cầu từ khách tới phục vụ file chứa thông tin cần thiết để truy nhập file.

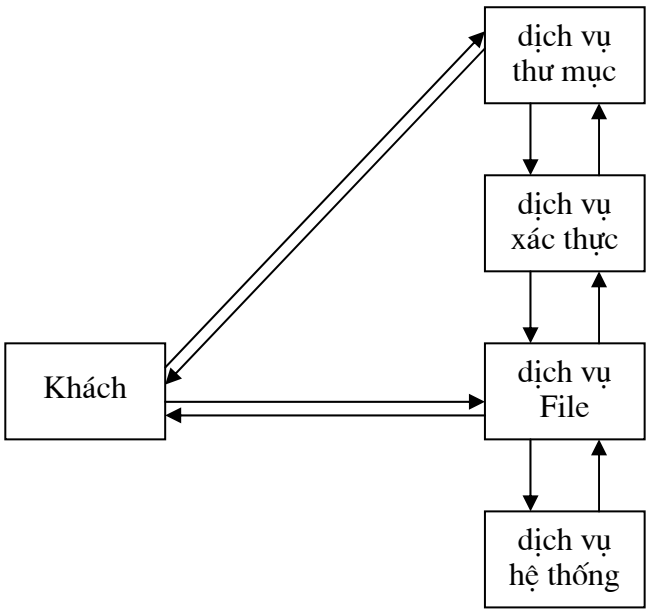
Các dịch vụ thư mục, xác thực và file là giao diện người dùng tới hệ thống file. Dịch vụ hệ thống là giao diện hệ thống file tới phần cứng và cần trong suốt tới người dùng. Dịch vụ hệ thống cung cấp các chức năng chính yếu gồm ánh xạ địa chỉ lôgic tới địa chỉ khối vật lý, tương tác tới các dịch vụ mức thiết bị đề định vị/giải phóng không gian file và thao tác file read/write thực sự. Dịch vụ hệ thống được hỗ trợ nhờ lời gọi hệ thống tới nhân. Cuối cùng, file trong hệ thống file có thể được cache nhằm nâng cao hiệu năng và được nhân bản để tăng tính tin cậy. Điều quan trọng là quản lý cache và nhân bản trở thành các dịch vụ hệ thống bản chất khác của hệ thống file. Cache và nhân bản là phức tạp hơn khi chia xẻ file trong hệ phân tán. Hai vấn đề quan trọng này trong thiết kế hệ phân tán được bàn luận trong phần sau.

6.2.2 Phuc vu và dich vu

Cần phân biệt dịch vụ với phục vụ. Phục vụ là quá trình thi hành dịch vụ. Một dịch vụ được thực hiện bởi phục vụ riêng hay nhiều phục vụ cộng tác nhau. Thông thường, hệ thống file lớn phân bố các file được quản lý tới các phục vụ file. Một phục vụ cũng

cung cấp dịch vụ đa thành phần, chẳng hạn như phục vụ thư mục thực hiện cả dịch vụ thư mục và dịch vụ xác thực.

Mối quan hệ Client/Server chỉ là tương đối. Để truy cập file, QT người dùng trước hết là QT khách với phục vụ thư mục. Phục vụ thư mục yêu cầu dịch vụ file và trở thành khách của phục vụ file. Việc xác thực truy nhập làm cho phục vụ file hoặc phục vụ thư mục trở thành khách tới phục vụ xác thực. Phục vụ xác thực có thể đòi hỏi dịch vụ của phục vụ file. Dịch vụ file dựa vào dịch vụ hệ thống tới các chức năng mức thấp do nhân cung cấp. Hình 6.3. cho ví dụ tích hợp giữa bốn dịch vụ này trong hệ thống file. Đường nối trong hình vẽ chỉ dịch vụ xác thực có thể trộn hoặc với phục vụ thư mục hoặc phục vụ file.



Hình 6.3. Tương tác các dịch vụ trong DFS

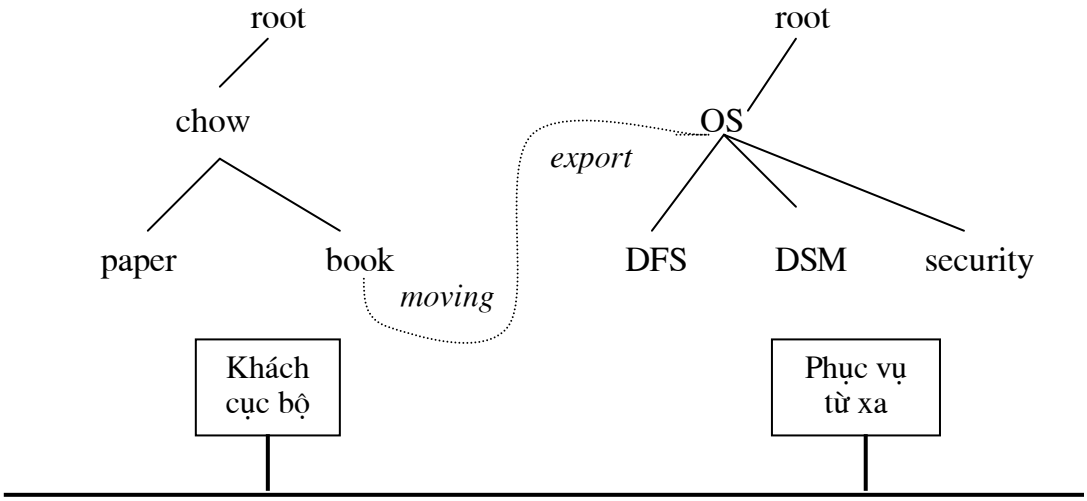
Thông thường, phục vụ phù hợp với máy tính cung cấp tài nguyên mà phục vụ quản lý. Phần lớn các hệ thống file tập trung lớn sử dụng các máy dành riêng như phục vụ file vì lý do hiệu năng và quản lý. Trong hệ thống như vậy, có sự phân biệt mang tính đặc trưng giữa máy phục vụ với máy khách. Tuy nhiên, trong môi trường phân tán các file là Server rải rác giữa một số máy tính, để hiệu quả hơn về giá thành thì mỗi máy tính đảm nhận cả hai vai trò khách và phục vụ.

Kiên định mô hình Client/Server và phân chia trách nhiệm trong hệ phân tán. các phục vụ trong DFS thường được cấu trúc nhằm cung cấp các dịch vụ thư mục, xác thực, file và hệ thống một cách riêng rẽ.

6.2.3. Gắn kết file và đăng kí phuc vu

Giả sử thư mục được cấu trúc phân cấp. "Gắn kết file" là khái niệm hữu dụng để xây dựng hệ thống file lớn từ các phục vụ file và thiết bị lưu giữ khác nhau. Thao tác gắn file do một khách thực hiện việc gắn một hệ thống file có tên từ xa tới hệ thống file khách phân cấp tại một điểm được định vị bằng một tên đường dẫn. Điểm gắn kết thường là lá của cây thư mục chứa tương ứng một thư mục con rỗng. Hệ thống file có tên được gắn kết được định danh bằng một máy tính từ xa hoặc một thiết bị cục bộ tiếp theo là tên đường dẫn tới hệ thống file. Hình 6.4. cho thấy hệ thống file được tạo thành nhờ gắn hệ thống file cục bộ với hệ thống file từ xa. Trong hình, hệ thống file từ xa "xuất khẩu" hệ thống file con OS chứa ba file DFS, DSM và ***Security***. Hệ thống file con này được gắn kết tới thư mục con của khách, ***book***, khi dùng giao thức gắn kết.

Mỗi lần file đã được gắn, chúng được truy nhập theo tên đường dẫn lôgic nối tiếp nhau không cần chỉ dẫn máy từ xa hay thiết bị cục bộ. Tính trong suốt định vị của các file được thi hành. Thông tin liên kết (thường được gọi là bảng gắn kết để định hướng lại truy nhập) đối với hệ thống file từ xa được hệ thống file của máy tính duy trì cho đến khi tháo gắn kết. Dùng cơ chế gắn kết như vậy, các khách khác nhau có thể nhìn theo khung nhìn hệ thống file khác nhau do các file từ xa có thể được gắn trên các thư mục con khác nhau của mỗi khách. Khi mong muốn có được khung nhìn hệ thống file toàn cục đồng nhất, quản trị hệ thống file tuân theo các luật gắn kết. Ví dụ, các file hệ thống chia xẻ bắt buộc gắn tới vị trí đã thoả thuận nào đó trong cấu trúc thư mục tách rời với hệ thống file cục bộ. Vì lý do an ninh, một phục vụ file thường chính xác gắn toàn bộ hoặc một phần hệ thống file của nó tới tập các máy đã được xác định trước. Thông tin này được chứa trong file ***export*** của phục vụ.



Hình 6.4.Gắn kết một hệ thống file

Gắn kết hệ thống file là thao tác đặc quyền và được thi hành theo ba thể hiện khác nhau:

(1) *Gắn hiển*: Khách đưa ra lời gọi gắn kết rõ ràng mỗi khi cần,

(2) *Gắn khởi động*: Một tập phục vụ file được quy định và tất cả các gắn kết được thực hiện tại thời điểm khởi động máy khách,

(3) *Gắn tự động*: Gắn các phục vụ được thực hiện tự động hoàn toàn theo yêu cầu trong lần đầu tiên file được khách mở.

Gắn hiển hệ thống file có tính linh hoạt (cho khách) nhưng tính quản lý kém hơn so với gắn khởi động, trong đó cấu hình đồng nhất của hệ thống file có hiệu lực tại thời điểm khởi động. Khi dùng gắn khởi động, khách có khung nhìn hệ thống file tĩnh song đầy đủ trong toàn bộ phiên tính toán. Lẽ tự nhiên, gắn khởi động lại có nhược điểm về phí tổn khi gắn các phục vụ không cần thiết. Gắn tự động kết hợp gắn hiển và gắn khởi động và có các ưu điểm của cả hai. Nó là động, giống như gắn hiển (chính xác hơn là file được gắn chỉ khi nó được truy nhập), và trong suốt, giống như gắn khởi động khi bỏ qua lời gọi gắn hiển. Khi một file được mở, nhân chọn một phục vụ từ tập phục vụ đã quy định sẵn và gắn hệ thống file được yêu cầu. Gắn tự động trong hệ thống file mạng (NFS) của Sun tiến hành thêm một bước nữa. Nó cho phép khách đặc tả số lượng phục vụ cùng hỗ trợ một dịch vụ file. Hệ điều hành quảng bá yêu cầu đó và gắn phục vụ đầu tiên đáp ứng được yêu cầu.

Gắn file là cách thuận tiện để đạt được tính trong suốt định vị truy nhập file. Tuy nhiên, gắn khởi động đòi hỏi tri thức định vị phục vụ file, vì điều đó, giao thức gắn là không trong suốt. Trong hệ thống có nhiều phục vụ file cung cấp cùng một dịch vụ file, thông tin định vị không còn thích hợp với khách. Phục vụ file có thể được định vị theo hai cách:

(1) Phục vụ file ghi nhận dịch vụ của nó bằng một *dịch vụ ghi nhận* và khách tra cứu *phục vụ ghi nhận* trước khi gắn,

(2) Khách quảng bá yêu cầu gắn và phục vụ file đáp ứng yêu cầu của khách.

Cả hai tiếp cận về cơ bản là giao thức giải pháp tên hoặc giải pháp địa chỉ. Đối với DFS, dịch vụ ghi nhận cung cấp lợi thế về hiệu năng. Phục vụ ghi nhận chỉ dẫn tới (đáp ứng) phục vụ file tốt nhất, hoặc phục vụ file làm cân bằng tải hệ thống cho yêu cầu của khách dựa trên tri thức của phục vụ ghi nhận về hệ thống. Phục vụ ghi nhận ứng xử giống như một ***thương gia*** thông minh trong hệ tự trị cộng tác. Khi dùng chính sách hợp lý, khách cũng có thể chọn một trả lời nhận được qua quảng bá. Tuy nhiên, thông tin kết nối thường trực trong nhân khách có thể được xem xét như cache. Nếu phục vụ file chuyển dời sang máy tính khác, truy nhập file tới phục vụ sẽ thiếu vắng cache. Một trong hai giao thức giải pháp được kích hoạt để tìm định vị mới của phục vụ bỏ qua sự can thiệp của khách. Kết quả là hoàn thành được tính độc lập định vị file.

Gắn hệ thống file từ xa thường được xếp chồng nhau. Hệ thống file được gắn có thể có thư mục nào đó được gắn từ một phục vụ file khác. Trong trường hợp đó, lặp lại tên đường dẫn đòi hỏi dọc theo bảng gắn đa thành phần trải trên phạm vi các máy tính. Khả năng phục vụ file đích sẵn sàng cho truy nhập song vấn đề là phục vụ file trung gian lại ngăn cản khách vươn tới file. Dùng giải pháp cho phép truy nhập trực tiếp tới file mỗi khi file được định vị. Thông tin này có thể được lưu trong cache đối với mọi file từ xa được mở. Giải pháp địa chỉ dựa trên đường dẫn mạng thích hợp hơn so với trên hệ thống file.

6.2.4. Phục vụ file có trạng thái và thiếu trạng thái

Phiên dãy thao tác file giống như kết nối dãy hỏi/đáp giữa khách và phục vụ file. Kết nối về bản chất là yêu cầu thiết lập và kết thúc một phiên truyền thông. Hơn nữa, tồn tại thông tin trạng thái tương ứng với mỗi phiên file (kết nối). Một số thông tin trạng thái điển hình là:

- Các file được mở và khách của chúng,
- Đặc tả file và thẻ file,
- Con trỏ định vị file hiện thời,
- Thông tin gắn,
- Tình trạng khóa,
- Khóa phiên,
- Cache hoặc buffer.

Thông tin trạng thái được duy trì động trong khi file đang được sử dụng. Chúng khác với thuộc tính file (thể hiện tính chất tĩnh của file). Vì lý do gì và khi nào thông tin trạng thái là cần đến? Bắt đầu từ thông tin đầu tiên trong danh sách trên, trạng thái buộc phải chứa file nào đang được mở và khách nào mở. Thông tin này được thường trực hoặc tại khách hoặc tại phục vụ hoặc cả hai. Tiếp theo, nhằm tránh lặp việc gọi dịch vụ thư mục cho mỗi lần truy nhập file, một đặc tả file và thẻ file của nó được cho đối với mỗi file mở và duy trì tại nhân của khách. Đối với những lần truy nhập tiếp theo, một con trỏ định vị file ẩn tương ứng với mỗi phiên file mở. Khi một file chia xẻ

được mở bởi nhiều khách, mỗi phiên file mở có con trỏ định vị file hiện thời của nó, ngoại trừ trường hợp thừa kế. Trong những trường hợp này, clinet thừa kế file từ cha của nó và con trỏ định vị file được chia xẻ cho QT cha và con. Con trỏ định vị file được bảo quản cùng với thông tin về file mở. Kế tiếp trong danh sách là thông tin gắn. Nhân của khách duy trì thông tin kết nối tới file ở xa. Phục vụ không cần biết file được gắn bởi khách nào mỗi khi file của nó được gắn.

Để điều phối chia xẻ file, thường muốn cho phép khách có năng lực khóa file. Do khóa là thông tin toàn cục chia xẻ bởi các khách, hợp lý là phục vụ file quản lý thông tin này. Một ví dụ khác về chia xẻ thông tin là khóa phiên trong truyền thông. Tồn tại một số khái niệm khi dùng khóa phiên để truyền thông an toàn giữa khách và phục vụ file. Khi xem xét trạng thái của một kết nối giữa khách với phục vụ file, đơn giản giả thiết khóa phiên là một bí mật thỏa thuận giữa phục vụ file và khách của nó, và khóa là thông tin trạng thái. Một số giao thức truyền thông an toàn hiện có dùng khóa phiên một lần. Trong tình huống này thì không cần bảo quản khóa như phần thông tin trạng thái. Cuối cùng, cache và buffer có thể được dùng hoặc tại khách hoặc phục vụ nhằm rút ngắn độ trễ. Thi hành cache và buffer là hữu dụng khi yêu cầu truy nhập đưa ra từ các QT cục bộ. Dữ liệu trong cache và buffer là thông tin trạng thái của kết nối.

Rõ ràng là thông tin trạng thái của phiên file có thể được phân tán giữa khách và phục vụ file truyền thông. Việc phân chia thông tin trạng thái có ảnh hưởng tới hiệu năng và quản lý của hệ thống file phân tán. Rõ ràng là, khách đòi hỏi duy trì những thông tin này. Cái gì mà phục vụ file cần đến đối với một phiên mở để phiên được xác định. Một phục vụ file được gọi là *đủ trạng thái* nếu nó duy trì thông tin trạng thái nào đó và được gọi là *thiếu trạng thái* nếu nó không duy trì một thông tin nào. Khi càng biết nhiều về kết nối thì càng mềm dẻo khi thực hiện điều khiển trên nó. Tuy nhiên, ý niệm về trạng thái là cái mà cần tránh xa càng nhiều càng hợp lý trong thiết kế hệ phân tán. Phục vụ file thiếu trạng thái là dễ dàng thi hành hơn và là thứ lỗi hơn theo hỏng hóc khách và chủ so với việc lưu giữ vết của thông tin trạng thái. Sự thỏa hiệp nằm giữa mềm dẻo và đơn giản. Do hỏng hóc là thường xuyên hơn trong hệ phân tán, nhiều thi hành DFS chọn thi hành dịch vụ file thiếu trạng thái. Hỏng hóc khách không ảnh hưởng tới phục vụ file. Sụp đổ của dịch vụ file thiếu trạng thái dễ khôi phục lại và được các khách khác nhận biết như là chậm trễ trả lời (hoặc không có trả lời) mà không cần phá vỡ phiên của chúng. Đối với dịch vụ file thiếu trạng thái, mỗi yêu cầu file tới phục vụ cần chứa thông tin đầy đủ về thẻ file, con trỏ định vị, khóa phiên, và các thông tin cần thiết khác. Thi hành phục vụ file thiếu trạng thái cần nhắm tới các vấn đề sau đây:

(1) Nhu cầu bất biến: Phục vụ file làm gì khi khách lặp hợp lý một yêu cầu do hỏng hóc trước đây (không nhận được trả lời) của phục vụ ? Thực tế hay không khi cấu trúc mọi truy nhập file như thao tác bất biến ?

(2) Cơ chế khóa file: Làm cách nào các khách chia xẻ thi hành cơ chế khóa file dựa trên phục vụ file thiếu trạng thái ? Nên chăng cơ chế khóa file được tích hợp vào dịch vụ giao dịch ?

(3) Quản lý khóa phiên: Làm cách nào để khóa phiên được sinh và được duy trì giữa khách và phục vụ file thiếu trạng thái truyền thông ? Có thể dùng khóa phiên một-lần cho mỗi lần truy nhập file ?

(4) Tính nhất quán cache: Phục vụ file có phải chịu trách nhiệm điều khiển tính nhất quán cache giữa các khách ? Ngữ nghĩa chia xẻ nào được hỗ trợ ?

Những biện luận trên đây chỉ ra rằng, về khái niệm phục vụ file thiếu trạng thái là đơn giản và hấp dẫn, thì trong thi hành thực tế một vài thông tin trạng thái tối thiểu cần được phục vụ duy trì. Nhu cầu bất biến và vấn đề khóa phiên chia xẻ đã được chỉ ra

trong chương 4 khi bàn luận về thi hành RPC. Giải pháp này được áp dụng tới phục vụ file nếu lời gọi thủ tục từ xa được dùng để yêu cầu dịch vụ file. Khóa file và nhất quán cache được bàn luận tiếp.

6.2.5 Truy cập file và ngữ nghĩa của chia xẻ

Độ phức tạp thi hành truy nhập đồng thời tới file chia xẻ từ xa phụ thuộc vào cách file được chia xẻ như thế nào và ngữ nghĩa của chia xẻ file được xác định từ người dùng.

a. Truy nhập file

Chia xẻ file có nghĩa rằng đa khách có thể truy nhập cùng một file tại cùng một thời điểm. Chia xẻ là kết quả của các thao tác truy nhập hoặc gối nhau hoặc xen kẽ nhau.

Truy cập gối nhau ngụ ý rằng có nhiều bản sao của cùng một file và đồng thời được thực hiện bởi *đa thành phần không gian* của file. Một ví dụ về nơi dữ liệu của file có thể được nhân bản là việc dùng cache. Để thuận lợi đưa bộ phận của file từ xa về cache cục bộ để truy nhập nhanh. Tương tự, đôi khi cần duy trì bản sao làm việc của một file trước khi một thay đổi lâu dài được làm trên file đó. Trong cả hai tình huống, bản sao nhân bản của file dữ liệu chia xẻ tồn tại và truy nhập đồng thời tới file trở thành thực hiện được. Mỗi vị trí có bản sao của file được nhân bản cho khung nhìn khác nhau đối với dữ liệu. Quản lý truy nhập tới bản sao cung cấp khung nhìn kết cấu của file được chia xẻ được gọi là *điều khiển kết cấu* (coherency control).

Truy nhập xen kẽ nhau do tính đa hạt thao tác truy nhập dữ liệu. Một vài truy nhập file đơn giản là thao tác đọc/ghi. Truy nhập khác lại bao gồm dãy các thao tác xen kẽ nhau trải dọc theo khoảng thời gian người dùng định sẵn, chẳng hạn như giao dịch hoặc phiên. Thực hiện xen kẽ nhau trên cùng một file cũng tạo ra một ảo giác về chia xẻ file đồng thời. Tính đồng thời được thực hiện bởi tính đa thành phần theo *thời gian* của file. Vấn đề quản lý ở đây là làm cách nào định trước ảnh hưởng một dãy thực hiện từ các dãy khác khi chúng xen kẽ nhau và làm cách nào ngăn ngừa kết quả không nhất quán hoặc có lỗi. Bài toán này thường được nói đến như là bài toán *điều khiển đồng thời* (concurrency control).

Không gian / *Thời gian*	*Truy nhập từ xa*	*Truy nhập cache*	*Truy nhập tải lên/xuống*
RW đơn giản	Không chia xẻ thực sự	Điều khiển nhất quán	Điều khiển nhất quán
Giao dịch	Điều khiển đồng thời	Nhất quán và đồng thời	Nhất quán và đồng thời
Phiên	Không thích hợp	Không thích hợp	Bỏ qua chia xẻ

Hình 6.5. Tính đồng thời không gian và thời gian truy nhập file

Hình 6.5. cho tính đa thành phần không gian và thời gian nhằm thực hiện hiệu quả việc truy nhập đồng thời tới file chia xẻ. Mục trong bảng là chức năng quản lý được yêu cầu đối với mỗi giả thiết chia xẻ.

Trong ***phạm vi không gian***, truy nhập đọc và ghi được thi hành theo một trong các cách sau, tùy thuộc vào dữ liệu trong file được trình diễn cho truy nhập như thế nào:

(1) *Truy nhập từ xa*: Không có dữ liệu file được giữ trong máy khách. Mỗi yêu cầu truy nhập được truyền trực tiếp tới phục vụ file ở xa, thông qua mạng hạ tầng.

(2) *Truy nhập cahce*: phần nhỏ của dữ liệu file được duy trì trong cache cục bộ. Thao tác ghi hoặc vắng cache tạo ra một truy nhập từ xa và cập nhật lại cache đó.

(3) *Truy nhập tải lên/tải xuống*: Toàn bộ file được tải xuống cho truy cập cục bộ. Truy nhập từ xa hoặc tải lên được thực hiện khi cập nhật file từ xa.

Tiếp cận đầu tiên là mô hình truy nhập đơn mức, còn các tiếp cận khác là trình bày bảo quản phân cấp hai mức đối với dữ liệu. Thi hành mô hình truy nhập từ xa đơn mức là trực tiếp do chỉ một bản sao của dữ liệu là tồn tại, và mọi yêu cầu truy nhập là tuần tự bởi phục vụ file. Mặt hạn chế nguyên thủy là độ trễ truy nhập mạng dài và hạn chế tính đồng thời. Mô hình cache và tải lên/tải xuống cho phép tính đồng thời và rút ngắn trễ truy nhập bằng cách giữ bộ phận / toàn bộ truy nhập là cục bộ. Tuy nhiên, tiệm cận này đưa đến bài toán kết cấu file. Do dữ liệu trong file được nhân bản và được chia xẻ cho một số người dùng đồng thời, file chia xẻ trình diện có thể khác nhau với những khách đồng thời khác nhau. Điều khiển kết cấu đối với bản sao dữ liệu là một bài toán không tầm thường. Kết cấu của dữ liệu nhân bản có thể được giải thích theo nhiều cách khác nhau. Những giải thích (định nghĩa) dưới đây được liệt kê theo một thứ tự nghiêm ngặt:

(1) Mọi bản sao là đồng nhất tại mọi thời điểm,

(2) Các bản sao được định sẵn đồng nhất tại chỉ một số thời điểm nhất định,

(3) Người dùng luôn đọc được dữ liệu "hiện thời nhất" trong bản sao,

(4) Thao tác ghi luôn được thực hiện "ngay lập tức" và kết quả của nó được truyền bá theo cách cố gắng nhất.

Định nghĩa (1) về tính kết cấu là lý tưởng song không thể thực hiện trong hệ phân tán do trễ truyền thống đáng kể. Bởi vậy, chấp nhận thực tế là không phải mọi bản sao là đồng nhất tại mọi thời điểm. Định nghĩa (2) là sự thỏa hiệp song lại khó xác định điểm đồng bộ tốt. Sự lưu hành dữ liệu là có ý nghĩa chỉ cần tại thời điểm đọc chúng. Như vậy, định nghĩa (3) có vẻ lôgic. Vướng mắc ở chỗ tính phỏng chừng của thuật ngữ, ***hiện thời nhất***, (hoặc ***muộn nhất***). Thứ tự tương đối của các thao tác ghi cần dựa trên thời điểm "hoàn thiện" chúng, là thời điểm kết quả của thao tác ghi đó phản ánh trong mọi bản sao. Một giao thức cần đến một thao tác đọc luôn phải cho kết quả của thao tác ghi hiện thời nhất. Định nghĩa (4) là một cố gắng thô sơ nhằm xấp xỉ định nghĩa (3).

Bài toán kết cấu file là thích đáng cho sự tồn tại của dữ liệu nhân bản và cập nhật bị làm trễ của khách tới bản sao. Kết cấu là bài toán hệ trọng hơn trong hệ phân tán do trễ mạng là dài hơn và kém dự báo hơn. Sự phức tạp của vấn đề trễ là nhân tố mà hỏng hóc (khách, phục vụ hoặc truyền thông) xảy ra thường xuyên hơn trong hệ phân tán. Hỏng hóc trong khi cập nhật file có thể đưa file về trạng thái không nhất quán. Một mong muốn cao đối với hệ thống đảm bảo tính nguyên tử của cập nhạt, chẳng hạn mỗi thao tác cập nhật hoặc hoàn thiện thành công hoặc không có kết quả nào. Với đa bản sao của cùng một file, một cập nhật file là nguyên tử nếu tất cả hoặc không bản sao nào được cập nhật. Cập nhật nguyên tử đòi hỏi hệ thống là *thu hồi được* (recoverable) (tức là, có khả năng hủy -undo- một cập nhật từng phần và phục hồi lại hệ thống trở về trạng thái trước khi cập nhật từng phần). Thao tác hủy bao hàm việc dùng lưu giữ trạng thái bền vững (stable) mà có thể được dùng khi khôi phục trạng thái hệ thống trong việc ứng phó với mọi hỏng hóc hệ thống. Tính nguyên tử của cập nhật nguyên tử của nhận bản được giới thiệu trong ngữ cảnh truyền thông đa tán phát nguyên tử. Điều này được xem xét thêm theo khung nhìn dữ liệu và giao thức quản lý nhân bản.

Trong ***pham vi thời gian***, các thao tác file có liên quan có thể được nhóm lại theo những đoan thời gian khác nhau tùy theo nhu cầu áp dụng chúng. Các thao tác đọc và ghi từ các QT khác nhau dẫn tới truy nhập file đồng thời. Một số giả thiết về truy nhập file là:

(1) *RW đơn giản*: Mỗi thao tác đọc và ghi là một truy nhập hỏi/đáp độc lập tới phục vụ file;

(2) *Giao dịch*: Một dãy các thao tác đọc và ghi được xem xét như là một đơn vị cơ sở truy nhập file (với một số đòi hỏi ngữ nghĩa bổ sung cho giao dịch). Về cú pháp, giao dịch được biểu diễn bởi một ***begin transaction*** tiếp theo là một số các truy nhập file và kết thúc với ***end transaction***. Giao dịch ở đây được định nghĩa theo nghĩa hẹp là các thao tác đọc và ghi truy nhập tới cùng một file.

(3) *Phiên*: Phiên chứa một dãy các giao dịch và thao tác RW đơn giản với ngữ nghĩa bổ sung phù hợp với phiên được định nghĩa bởi ứng dụng. Điển hình, phiên được gói gọn trong cặp các thao tác mở và đóng file.

Truy nhập RW đơn giản và phiên là các khái niệm quen thuộc trong mọi hệ điều hành. Giao dịch là một khái niệm đồng bộ mức cao. Truy nhập dữ liệu của một giao dịch được chứa đựng trong cặp ***begin transaction/end transaction***. Dãy các truy nhập (bao gồm cập nhật) là nguyên tử hoặc ***không thể chia cắt*** với nghĩa dãy các thao tác được thực hiện bỏ qua sự giao thoa từ các khách khác. Kết quả của việc thực hiện xen kẽ các truy nhập bởi nhiều khách là tương đương với sự thực hiện tuần tự của dãy, đưa dữ liệu từ trạng thái nhất quán này tới trạng thái nhất quán khác. Nếu các tính chất nguyên tử, nhất quán, cô lập và bền vững (ACID) dược đảm bảo (xem truyền thông giao dịch), truy nhập dữ liệu được tạo trong một *giao dịch nguyên tử*. Khái niệm giao dịch nguyên tử là phổ dụng với những ứng dụng thương mại. Chức năng làm hiệu lức tính nguyên tử và nhất quán trong phục vụ file được gọi là *dịch vụ giao dịch*.

b. Ngữ nghĩa của chia xẻ

Chia xẻ file sử dụng truy nhập đa thành phần không gian và thời gian đưa tới baif toán điều khiển kết cấu và đồng thời. Giải pháp tới bài toán phụ thuộc vào ngữ nghĩa của chia xẻ do các ứng dụng yeu cầu. Về lý tưởng, cần cập nhật một file là hoàn thiện ngay lập tức và kết quả của nó trở nên nhìn thấy được một cách tức thì đối với các QT chia xẻ khác. Về thực tế, ngữ nghĩa này là khó thực hiện và do đó thường là nhẹ nhàng hơn. Trễ của thao tác ghi là không thể tránh được trong hệ phân tán do kết quả của thao tác ghi có thể được trễ bởi mạng hoặc bởi dịch vụ hệ thống theo đòi hỏi kết cấu và nhất quán. Dùng quan điểm không gian và thời gian như hình 6.5, có danh sách ba mô hình phổ biến nhất cho các mục tiêu khác nhau của chia xẻ file:

(1) *Ngữ nghĩa UNIX*: Kết quả của thao tác ghi được lan truyền tới file và các bản sao của nó tức thì sao cho các thao tác đọc sẽ nhận được giá trị "muộn nhất" của file. Không bị chậm lệnh ghi phải chịu ngoại trừ trễ mạng tất yếu. Truy nhập tiếp theo của khách phát ra thao tác ghi buộc phải chờ thao tác ghi hoàn thiện. Mục tiêu nguyên thủy là duy trì tính hiện thời của dữ liệu.

Việc chia xẻ nội dung

Khi sử dụng file chung truy cập thời gian ảnh hưởng sự liên kết và điều khiển. Giải pháp cho vấn đề này phụ thuộc vào yêu cầu chia xẻ của ứng dụng. Thực tế chúng ta muốn cập nhật hoá một cách hoàn toàn tới file và có ngay kết quả của quả trình chia xẻ. Trường hợp đặc biệt nội dung này khó đạt được và luôn bị giảm xuống. Thời gian trễ là không tránh khỏi trong hệ phân tán vì quá trình ghi bị trì hoãn ảnh hưởng bởi mạng hoặc hệ thống dịch vụ khác cố định có liên kết với nó. Sử dụng khái niệm không gian và thời gian ta lập ba mô hình nội dung có thực thể khác nhau cho file chia xẻ:

Nội dung HĐH (UNIX): Kết quả quá trình ghi truyền tới file và được sao ra lập tức, thao tác đọc trở thành công việc mới nhất của file. Không có sự trễ nào trừ sự trì

hoãn của mạng là không tránh khỏi. Truy cập từ máy PC phải chờ quá trình ghi hoàn thành, khoá duy trì dữ liệu hiện thời.

Nội dung công việc: Kết quả quá trình ghi có thể không lưu trong vùng nhớ đang làm việc và chuyển đổi chỉ khi vài kết quả cố định bắt buộc để ở cuối quá trình. Đối tượng khoá duy trì dữ liệu cố định.

Nội dung đoạn: Ghi vào file thể hiện qua sự nhân bản, kết quả tạo ra cố định chỉ tại nơi kết thúc đoạn. Nội dung khoá duy trì sự truy cập dữ liệu có hiệu quả.

Khác biệt chính giữa nội dung HĐH và 2 loại khác là sự trễ trong quá trình ghi. Unix cho phép ghi ngay hiệu quả trong khi nội dung đoạn và công việc sử dụng độ trễ. Unix dễ thao tác với bộ xử lý đơn nơi mà dữ liệu được lưu trong các file cơ bản, file chung không được định danh và mọi thao tác đọcghi trực tiếp bởi file chủ. Nếu dữ liệu được lưu trong một quá trình xử lý cơ bản (thường trong hệ thống phân tán) nhiều bộ nhớ nhỏ chứa cùng file dữ liệu có thể tồn tại. Yêu cầu ghi vào file chủ xử dụng điều kiện kế hoạch lưu. Tuy vậy kết quả không đúng trừ khi bộ nhớ vừa được cập nhật hoá. Vấn đề cùng lưu bộ nhớ về logic phức tạp như vấn đề bộ nhớ cố định trong hệ thống bộ nhớ phân tán chia xẻ. Cách thức ghi đúng và cập nhật như thế nào được thảo luận ở chương sau.

Thông tin đúng trên file chủ lưu dữ liệu được cập nhật thường xuyên. Truy cập file bằng việc ghi vào bộ nhớ định danh mô hình nội dung. Thao tác ghi vào bộ nhớ không hiệu quả bằng thao tác đọc. Cải tiến quá trình ghi bằng cách ngừng ghi nếu mọi quá trình ghi vào file chủ không cần thiết ngay lập tức. Rất khó để quyết định quá trình ghi có ảnh hưởng tới quá trình không, giải pháp là ghi vào file chủ một cách có định ký, không ảnh hưởng tới quá trình xử lý khác. Chiến lược điều khiển ghi trực tiếp vào bộ nhớ lfm việc truy cập tăng lên không được duy trì chính xác trong nội dung UNIX.

HĐH cố gắng chia xẻ dữ liệu toàn bộ cho các máy PC, trong nhiều trường hợp nó giải quyết sự lãng phí. Vài ứng dụng không liên quan đến bản sao, dữ liệu cố định bị xâm phạm. Hệ thống chỉ đòi hỏi kết quả thực hiện cân bằng thao tác thực hiện đoạn. Việc cập nhật file cố định lâu hơn trong mô hình nội dung công việc. Mỗi khách chứa file sao từ file chủ có thể sửa đổi nội dung file sao tới hết đoạn file, khi file đóng sửa đổi file được sao từ file chủ. Tính chất này không được áp dụng nếu có nhiều khách. Việc chia xẻ hạn chế giữa nhiều khách sẽ dễ thực hiện và áp đáp ứng đầy đủ các yêu cầu ứng dụng.

6.2.6 Điều khiển phiên bản

Mô hình nội dung đoạn có nhiều hạn chế. Yêu cầu đóng đoạn ngay sau quá trình ghi tương đương mô hình nội dung HĐH với quá trinhf ghi vào bộ nhớ. Yêu cầu file mới được tạo bằng cách sửa đổi file cũ (file cũ chỉ cho phép đọc). Mọi vấn đề với file lúc này là chia xẻ và tạo mới vì nó chỉ cho phép đọc. Để cùng ghi, khách phải biết tên file mới tạo. Giải pháp đơn giản của vấn đề là sử dụng cùng tên file nhưng khác phiên bản. Chức năng dịch vụ đường dẫn thực hiện việc điều khiển phiên bản. Mỗi file trong đường dẫn tương ứng với một số. File có số hiệu phiên bản cao nhất là file hiện tại. Nếu một phiên bản hiện tại được truy cập bởi nhiều khách khác thì có ba cách thể hiện phụ thuộc vào dữ liệu trong file đang chia xẻ và sửa đổi như thế nào?

Lờ đi sự xung đột: Một phiên bản mới được tạo bất chấp điều gì xảy ra.

Phiên bản giải quyết lại sự xung đột: giả thiết dữ liệu thay đổi trong phiên bản không tương thích với phiên bản mới hiện tại. Kết quả việc cập nhật hoá có mâu thuẫn. Vấn đề này được giải quyết bằng việc cập nhật hoá phiên bản hiện tại với phiên bản mới (kết nối mọi cập nhật).

Giải quyết một chuỗi xung đột: giả sử dữ liệu đã được sửa đổi trong phiên bản hiện tại. Việc cùng cập nhật hoá có thể dẫn đến một chuỗi và chạm (mâu thuẫn). Giải pháp là lờ đi hoặc nhảy qua công việc của khách với một phiên bản mới đồng thời cập nhật theo trật tự tuỳ ý.

Nhiều xung đột phức tạp và điều khiển phiên bản có thể giải quyết phụ thuộc vào yêu cầu ứng dụng. Sử dụng file cố định không giải quyết vấn đề hơn là quan tâm tới dịch vụ file và các hệ thống dịch vụ khác. Để giảm yêu cầu bộ nhớ của nhiều tổ chức phiên bản, một phiên bản mới phải chứa tính chất thay đổi trường file (khối). Số hiệu phiên bản phụ thuộc sự thay đổi khối. Duy trì rãnh thay đổi khối dễ tạo sự xung đột cao.

6.3 Điều khiển đồng bộ và giao dịch

Điều khiển các giao dịch được phát triển cho vấn đề quản lý dữ liệu nhưng chúng ta có thể áp dụng cho việc quản lí hệ thống file phân tán. Trong phần này chúng ta thảo luận xem các giao dịch có thể thực hiện như thế nào trong hệ phân tán đặc biệt là trong hệ thống file phân tán.

6.3.1 Dịch vụ giao dịch

Hình 6.6 mô tả cấu trúc đơn giản của hệ xử lí giao dịch phân tán. Khách tại mỗi vị trí phân tán đưa ra yêu cầu dịch vụ giao dịch đối với hệ xử lí giao dịch cục bộ (TPS), hệ này gồm ba thành phần chính: quản lí giao dịch (TM), lập lịch (SCH), quản lí đối tượng (OM). Một giao dịch có thể thao tác trực tiếp trên những dữ liệu ở xa. Những thao tác này được thực hiện như những giao dịch phụ và được gửi tới những TPS ở xa. Việc quản lí những giao dịch liên quan tới nhứng TM khác xem như đó là sự thực hiện của cả giao dịch địa phương và cả những giao dịch phụ ở xa. Nhiệm vụ của quản lí kế hoạch là thao tác lập kế hoạch trên đối tượng để tránh xung đột. Việc quản lí đối tượng đảm bảo tính nhất quán cho việc nhân bản, lưu giữ và cung cấp giao diện cho hệ thống file. Chúng ta sử dụng cấu trúc đơn giản về dịch vụ giao dịch phân tán để địa chỉ hoá 3 yêu cầu cơ bản:

- Thực hiện tập các thao tác trong mỗi giao dịch là toàn bộ hoặc không.
- Thực hiện giao dịch xen kẽ thực hiện giao dịch khác là không bị chia cắt,
- Cập nhật đối tượng nhân bản là nguyên tử.

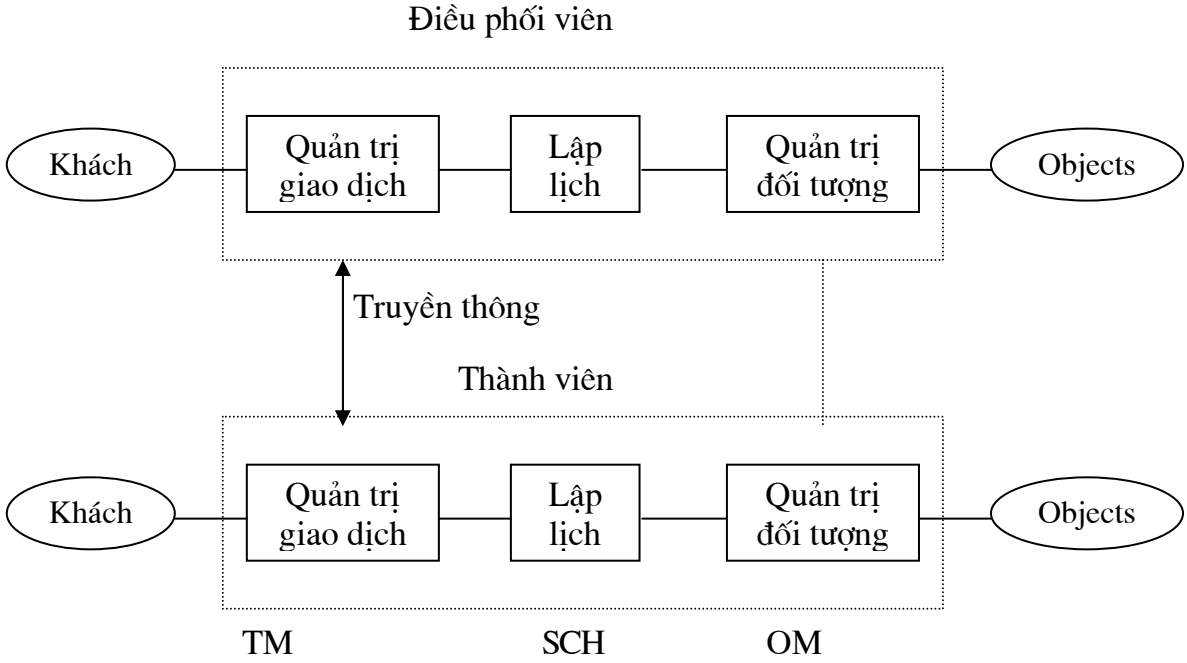
Giao thức thực hiện tất cả hoặc không và che giấu giao dịch phân tán thường phức tạp. Chúng ta có thể sử dụng hình 6.6 để minh hoạ như sự tiếp cận giải quyết. Việc tổng kết các chức năng này thể hiện qua 4 thực thể (xử lí khách, quản lí giao dịch, lập kế hoạch, quản lí đối tượng) và sự ảnh hưởng lẫn nhau giữa chúng.

Hệ thống xử lí giao dịch:

Xử lí đối với khách: một quá trình xử lí chỉ ra sự bắt đầu giao dịch bằng cách thông báo bắt đầu giao dịch tới *Bộ quản lí giao dịch địa phương* (TM). Khi nhận được yêu cầu trên TM sinh ra một giao dịch id và không gian làm việc cho khách với thao tác đọc ghi thường xuyên. Sự kết thúc giao dịch từ một khách tới TM chỉ ra khách đã sẵn sàng uỷ thác giao dịch và đang đợi TM để trở lại trạng thái uỷ thác ban đầu tại tất cả các TM có liên quan đến giao dịch này được xem như là việc uỷ thác hoặc dừng đã thành công. Giao thức được dùng chi tiết và cố định với quản lí giao dịch, kế hoạc và quản lý dữ liệu hoàn toàn dễ dàng với người dùng

Quản lý giao dịch (MT): MT tạo ra một giao dịch id và không gian làm việc thực hiện giao dịch khởi đầu của lient theo đó mỗi lần truy cập đòi hỏi một đối tượng dữ liệu đồng thời tạo ra số hiệu giao dịch id, gửi tới bộ quản lý kế hoạch. Kết quả truy cập trong không gian làm việc chỉ được nhận thấy bởi lient. Đây là pha thực hiện của giao

dịch tại TM. Quản lí giao dịch biết được những giao dịch địa phương và những giao



Hình 6.6. Hệ thống xử lý giao dịch phân tán

dịch phụ ở phía xa.

Truy cập từ TM được chấp nhận hoặc phản đối bởi bộ lập lịch (SCH).

Nếu chấp nhận sẽ thay đổi trạng thái của không gian làm việc và nội dung của lient để duy trì thuộc tính tất cả hoặc không phải có sự phản hồi tới TM để dừng thao tác bằng cách gửi đi một tín hiệu dừng với lient và tất cả bộ quản lý giao dịch khác có liên quan đến giao dịch đó nếu như không có tín hiệu dừng sẽ thực hiện hết giao dịch có nghĩa là tất cả quản lý giao dịch bao gồm có pha thực hiện, đọc, ghi và pha uỷ thác để quyết định xem giao dịch nào sẽ được uỷ thác và dùng. Điều này được thực hiện bởi giao thức uỷ thác 2 pha. Nếu như thành công thì việc cập nhật sẽ được thực hiện tại các OM và trạng thái được uỷ thác gửi tới lient, nếu không quản lý giao dịch sẽ dừng.

Lập lịch (SCH): SCH chọn ra giao thức điều khiển đồng bộ để đảm bảo cho sự thực hiện đồng bộ của các giao dịch tại các vị trí phân tán. Có 3 cách thức chính: ngăn ngừa, tránh, lỗi. Với cách thức ngăn ngừa tất cả các yêu cầu truy cập đối với một giao dịch có thể gây nên xung đột sẽ ngăn ngừa. Trách nhiệm TM là thay đổi giao dịch của trạng thái lient sang trạng thái ngừng, đặc biệt khoá được dùng cho cách thức này. Việc tránh thì bộ kế hoạch sẽ kiểm tra mỗi thao tác truy cập riêng rẽ xem thao tác nào sẽ được thực hiện. Cách thức phổ biến được dùng là tem thời gian. Bộ kế hoạch sẽ đưa ra quyết định dựa trên thứ tự của tem thời gian. Cách thức lỗi thì xung đột hoàn toàn không được quan tâm đến trong suốt quá trình thực hiện.

Quản lý đối tượng (OM): OM chịu trách nhiệm với các dịch vụ file về thao tác thực sự trên các dữ liệu, cung cấp quản lí bộ nhớ hiệu quả bằng giao thức phản hồi lỗi. Nếu đối tượng dữ liệu được nhân bản thì OM sẽ quản lí sự nhân bản bằng cách sử dụng giao thức quản lí.

6.3.2. Chuỗi thực hiện liên tiếp

Khái niệm giao dịch rất có ích đối với dữ liệu phức tạp trong hệ thống phân tán đa người dùng. Thuộc tính ACID đảm bảo rằng không có sự đối nghịch được gây nên bởi giao dịch khác hoặc lỗi hệ thống.

Mỗi giao dịch trong hệ thống dữ liệu bao gồm tuần tự các thao tác đọc ghi. Để tránh tình trạng hư hỏng dữ liệu phải khoá trước khi đọc, ghi. Hơn nữa thì thao tác đọc đồng bộ và thao tác ghi duy nhất phải đảm bảo khác biệt thì một khoá dungf chung được dùng cho đọc đồng thời và khoá duy nhất được dùng cho ghi. Tất cả các khoá phải được mở sau mỗi thao tác và trước khi kết thúc một thao tác. Nếu khoá không được dùng thao tác ghi có thể chỉ được thực hiện không dứt khoát.

Tất cả các thao tác trong tập hợp các giao dịch được sắp xếp thành lịch để thực hiện. Một lịch được gọi là liên tiếp nếu kết quả thực hiện diễn ra tuần tự theo lịch.

Giả sử có ba giao dịch : t0, t1, t2 đã được thực hiện, t1 và t2 là hai giao dịch khác cùng thực hiện.

t0: ***bt*** write A=100; write B=20; ***et***

t1: ***bt*** read A; read B; 1: write sum in C; 2: write diff in D ***et***

t2: ***bt*** read A; read B; 3: write sum in C; 4: write diff in D ***et***

Có 3 giao dịch chia xẻ đối tượng dữ liệu định vị ở những vị trí khác nhau như ở hình 6.6. Thao tác (1,3) và 2,4) đều ghi dẫn tới xung đột trên C,D; các thao tác khác không có xung đột. Có khả năng chèn vào thao tác 4.

Nếu t1, t2 thực hiện liên tiếp thì để hoàn thành (C,D) sẽ là (80,120) hoặc (120,80). Với Wi trên đối tượng tại C,D là thao tác ghi. Việc cập nhật sẽ được ghi nhớ và được ghi tại mỗi vị trí.

Bảng 6.1. Lập lịh xếp chèn

Lịch	Xếp chèn	Đăng nhập tại C	Đăng nhập tại D	Hàm Result (C,D)	2PL	Tem thời gian
1	1,2,3,4	$W_1 = 120$ $W_2 = 80$	$W_1 = 80$ $W_2 = 120$	(80,120) nhất quán	khả thi	khả thi
2	3,4,1,2	$W_1 = 80$ $W_2 = 120$	$W_1 = 120$ $W_2 = 80$	(120, 80) nhất quán	khả thi	t1 bỏ dở và bắt đầu lại
3	1,3,2,4	$W_1 = 120$ $W_2 = 80$	$W_1 = 80$ $W_2 = 120$	(80,120) nhất quán	không khả thi	khả thi
4	3,1,4,2	$W_1 = 80$ $W_2 = 120$	$W_1 = 120$ $W_2 = 80$	(80, 80) nhất quán	không khả thi	t1 bỏ dở và làm lại
5	1,3,4,2	$W_1 = 120$ $W_2 = 80$	$W_1 = 120$ $W_2 = 80$	(80, 80) không nhất quán	không khả thi	bỏ dở nhiều
6	3,1,2,4	$W_1 = 80$ $W_2 = 120$	$W_1 = 80$ $W_2 = 120$	(120,120) không nhất quán	không khả thi	t1 bỏ dở và bắt đầu lại

6.3.3. Giao thức điều khiển đồng bộ

Có một số phương pháp chung đối với vấn đề điều khiển đồng bộ nhờ vậy có thể những mâu thuẫn được ngăn ngừa hoặc tránh được hoặc tính tương thích có thể có lỗi. Trong phần này chúng ta thảo luận phương thức khoá 2 pha, tem thời gian và điều khiển đồng bộ tối ưu, đó là một số phương pháp.

Khoá 2 pha

- Sử dụng phương thức khoá có nghĩa các đối tượng bị chia xẻ đạt được tính trong suốt phải được khoá khi chúng có thể được truy cập và phải được tự do trước khi kết thúc thời gian truy cập. Khoá 2 pha thêm đòi hỏi một khoá mới không thể được yêu cầu sau khi lần mở đầu tiên của khoá. Một quá trình được chia thành 2 pha: một là

khoá đối tượng và một pha mở. Một trường hợp của phương thức khoá 2 pha là khoá tất cả các đối tượng dữ liệu ở thời điểm bắt đaàu của quá trình và mở khóa ở thời điểm kết thúc của quá trình. Chuỗi những quá trình lặp lại bởi chỉ có những quá trình nào thuận lợi mới tham gia thực hiện. Phương pháp được dùng đối với những dữ liệu đơn giản mà không phải cho tất cả vì nó không đáp ưngs được về vấn đề chia xẻ dữ liệu cũng như đồng bộ. Một bước cải tiến là mở khoá ngay trong những điều kiện có thể. Tuy nhiên cách thức vẫn được làm là mở khoá ngay tại thời điểm giao thời. Khoá 2 pha được thực hiện như chuỗi quá trình có thể được xem xét ở trên. Nếu quá trình t1 đòi hỏi đối tượng C để cho thao tác 1 đối tượng sẽ không được mở chỉ sau khi 2 đã hoàn thành. Khi đó có thể thao tác 3 tại thời điểm t2 xuất hiện giữa thao tác 1 và 2. Tương tự thời điểm t2 chứa khoá của đối tượng C đầu tiên thao tác 1 trong thời điểm t1 không thể được thực hiện giữa thao tác 3 và 4. Để cập nhật đối tượng D cũng tương tự như đối với C. Điều này có nghĩa danh mục 3,4,5 và 6 trong bảng 6.1 là không khả thi nếu t1 và t2 thực hiện theo phương thức 2PL. Danh mục kết quả được giới hạn đối với danh mục 1 và2. Vì vậy khoá 2 pha không đem lại tính đồng bộ cho xử lí chuối quá trình.

Khoá đối tượng này và đợi đối tượng khác là bế tắc trong bất kì hệ thống nào. Nếu thao tác 3 và 4 là được duy trì tại thời điểm t2 có thể tại thời điểm t1 có thể đang thao tác với đối tượng C và đợi D trong thời điểm t2 đang thao tác với D và đợi C. Kết quả là bế tắc đợi và lưu vòng tròn.

Dường như TM sẽ có những cách thức mở khoá ngay sau khi có thể. Ngay khi biết rằng khoá cuối cùng đã được mở và dữ liệu không thể được truy cập lần nữa. Tuy nhiên có nhiều những khó khăn đối với phương thức này. Trước hết chúng ta có thể gặp phải sự dừng chương trình vòng tròn. Cứ cho rằng thời điểm t1 xử lý với dữ liệu C và mở khoá đối với C, bước tiếp thời điểm t2 đọc giá trị trên C. Nếu thời điểm t1 dừng thì t2 cũng phải dừng bởi vì t2 đã đọc dữ liệu. Sự dừng t2 dừng t3 và các thời điểm khác. t2 có thể uỷ thác chỉ khi t1 uỷ thác, và t2 đã độc lập với t1. Sự độc lập về quyền uỷ thác phải được xem xét tức là sự uỷ thác của t2 phải đợi cho đến khi t1 thực hiện điều đó rồi.

Vấn đề trên có thể được giải quyết bằng việc sử dụng khoá 2 pha nghiêm ngặt. Một giai đoạn chỉ có thể được mở tại điểm uỷ thác hoặc tại thời điểm dừng. Với hệ thống khoá 2 pha khó có thể thực hiện do không nghiêm ngặt vì TM không biết khi nào thì khoá cuối cùng mở. Khoá 2 pha nghiêm ngặt chưa thật sự đem lại sự thực hiện một cách đồng bộ hoàn hảo song đã đơn giản hơn trong thực hiện.

Tem thời gian

Trình tự các thao tác ma khoá 2 pha đem lại đối với những đối tượng dữ liệu chia xẻ là từ khoá đầu tiên. Hiển nhiên chúng ta đã xem xét nhiều ứng dụng của tem thời gian đối với thứ tự sự kiện. Có những xung đột giữa các thời điểm cũng nảy sinh với các tem thời gian. Chúng ta giả sử rằng t0, t1, t2 ở ví dụ trước đại diện cho 3 thời điểm với t0<t1<t2 khi một thao tác xử lí trên đối tượng dữ liệu chia xẻ là được thực hiện, đối tượng ghi thời gian thực hiện. Cho rằng sau đó thì những quá trình khác cũng thao tác trên dữ liệu. Nếu quá trình có thời gian lớn hơn thì ta để cho quá trình đó thực hiện. Nếu QT có thời gian nhỏ hơn chúng ta phải dừng vì nếu không nó sẽ vượt quá thời gian. QT bị dừng sẽ bắt đầu lại thực hiện với thời gian lớn hơn.

Để thao tác một có tranh chấp tới QT xử lý có nghĩa thao tác được phép đợt đến lượt. QT với tem thời gian lớn hơn sẽ đợi nhỏ hơn và QT với tem thời gian nhỏ hơn dừng và thực hiện khi chúng tiến đến QT có thời gian lớn hơn. Sự bế tắc được cởi bỏ bởi sự thực hiện của các QT dựa trên sự tăng dần của các tem thời gian bao hàm khả năng xảy ra tình trạng lưu và đợi. Sử dụng phương thức tem thời gian danh mục 1 và 3 trong

bảng 6.1 là khả thi. Đối với danh mục 2,4 và 6 QT t1 là quá trình chậm để thao tác trên C. QT t1 bị dừng trước khi nó truy cập đến D. Với tem thời gian lớn hơn cũng sẽ được thực hiện tương tự như 1 và 3.

Với danh mục 5 thì phức tạp hơn. Thao tác 1,3 và 4 thực hiện không có vấn đề gì nhưng với 2 thao tác trên D trong QT t1 xem xét dựa trên tem thời gian lớn hơn được ghi bởi QT t2 thao tác trên D. Để dừng QT t1 chúng ta cũng phải vô hiệu hoá thao tác của nó trên C. Chẳng hạn đối tượng C xem như được thực hiện với t1 rồi t2 của thao tác 3 gây ra không hiệu quả đối với t2. Tuy nhiên, nếu thao tác s3 là đọc dữ liệu t2 sẽ đọc dữ liệu mà không ảnh hưởng tới nó. Do vậy t2 cũng sẽ bị dừng, Để tránh sự mất đi đặc tính độc lập, thao tác đọc và ghi phải được độc lập với nhau và thao tác ghi chỉ có thể chậm hơn trước thời điểm uỷ thác để đảm bảo tính độc lập. Thao tác ghi được duy trì bởi danh mục SCH.

Để thực hiện giao thức với thao tác ghi chậm mỗi đối tượng được gắn kết với một tem thời gian logic chỉ cho thấy rằng thời gian do dự được rút ngắn dần theo thứ tự giảm dần và Tmin là thời gian nhỏ nhất. Không cần thiết cho thời gian đọc do dự nhưng mỗi đối tượng phải được lưu giữ theo mỗi QT đọc coót để khi có một sự uỷ thác thời điểm sớm nhất có thể được ghi như là RD mới.

Hình 6.7 cho thấy diễn biến tuần tự diễn ra. Giao thức uỷ thác 2 pha thứ tự theo gói thời gian chỉ với WR và Tmin được mô tả các QT đọc, ghi, dừng và uỷ thác

Đọc: Thao tác đọc này không ảnh hưởng tới thao tác đọc khác. Vì vậy thời gian của nó được so sánh với WR và Tmin. Thao tác bị dừng để duy trì cho thứ tự gói thời gian tăng dần nếu thời gian của nó nhỏ hơn Wr. Nó cho phép xử lí với thời gian lớn hơn WR nhưng nhỏ hơn Tmin. QT đọc được khởi tạo lại trong không gian làm việc của TM và quay về máy Khách. Một đối tượng có thể thực hiện thao tác ghi lưỡng lự khi thao tác đọc thực hiện (thời gian của thao tác đọc lơn hơn Tmin). Thao tác đọc được nhận thấy trong danh sách và đợi cho đến khi thao tác ghi uỷ thác cho nó.

Ghi: một thao tác ghi ảnh hưởng tới cả đọc và ghi. Vì vậy nó chỉ được phép do dự khi gói thời gian lớn hơn cả RD và WR. Thao tác do dự này được lưu vào trong danh sách do dự. Điều hành các QT được xem là thành công hay thất bại của các thao tác ghi trước.

Dừng: Việc dừng thao tác đọc không ảnh hưởng tới QT khác OM được xem như là đợi đọc. Nếu việc ghi dừng để ghi vào danh sách do dự. Nếu đến đỉnh của danh sách thì thao tác đọc được thực hiện.

Uỷ thác: Sự uỷ thác chỉ được xem xét chỉ sau khi sự thành công của sự uỷ thác trong TM. Sự uỷ thác cho một QT đợi để đọc có thể không bao giờ xảy ra vì QT bị kết khối. Đối với quá trình uỷ thác chỉ liên quan đến thao tác đọc OM ghi thời gian thực hiện của RD như là RD mới. Nếu QT uỷ thác đã thực hiện một thao tác ghi SCH dừng tất cả các thao tác (cả đợi đọc và ghi) đầu của QT được uỷ thác kéo dài quá trình cập nhật và loại bỏ khỏi danh sách ngừng trệ. Việc này cho phép thao tác đọc đang đợi tiến hành. Cập nhật thường xuyên gọi là sự điều khiển định danh nếu bản sao của những đối tượng tồn tại.

Điểm chú ý cuối cùng của bài toán điều khiển đồng bộ theo phương thức gói thời gian là về về tính phức tạp vì có nhiều QT chạy ở cùng một thời điểm. Giao thức uỷ thác sử dụng ví dụ này như là dẫn chứng có thể dẫn đến bế tắc khi TM không thể tiếp nhận hầu hết những điều kiện cần thiết từ TM khác. Gói thời gian có thể được dùng để giải quyết những vấn đề liên quan đến uỷ thác.

Điều khiển đồng thời tối ưu

Điều khiển đồng thời (ĐKĐT) dùng khoá là không tối ưu do xuất hiện những chuỗi những sự kiện. Điều khiển theo tem thời gian thì có phần tối ưu và cho phép các QT thực hiện dễ dàng. Có xuất hiện kết khối vì vậy một số đòi hỏi phải được đáp ứng. Chúng có thể cải tiến để cho phép toàn bộ QT được hoàn thành và làm cho những QT này trở nên có hiệu quả trước khi có những ảnh hưởng lâu dài. Giả thiết rằng khoảng rỗi chứa những phiên bản của dữ liệu chia xẻ có sẵn ở mỗi QT. Vì phương thức này cho thấy xung đột ít xuất hiện có thể không xuất hiện nên nó được gọi là điều khiển đồng bộ tối ưu.

Với 2PL và tem thời gian tác hoạt được chia thành 2 giai đoạn là pha thực hiện và pha uỷ thác. Hệ thống xử lí các QT bắt buộc hoặc giải quyết tính tương thích trong giai đoạn thực hiện và làm cho chúng thành nhân tố cơ bản trong giai đoạn sau. Đối với điều kiện bài toán đồng bộ tối ưu sẽ không có tính nhất quán cho đến khi kết thúc QT. Tại thời điểm TM bắt đầu pha hợp thức. Một QT được hợp thức toàn bộ giữa thời điểm TM bằng việc dùng giao thức uỷ thác 3 pha. Sau khi hoàn thành thì TM chuyển sang pha cập nhật để thực hiện một số htay đổi tổng bộ nhớ. Vì điều đó không có nghĩa đối với những QT đã được hoàn thành chúng ta giả sử rằng với mỗi tiến tình đã được hợp thức nó phải đảm bảo đã được uỷ thác. Điều này có nghĩa tất cả mọi sự uỷ thác đều phải thực hiện sau thời điểm hợp thức. Vì vậy pha cập nhật phải là cơ bản. Giao thức điều khiển tối ưu có 3 pha: thực hiện, hợp thức và tiếp cận. Mỗi QT ti được gắn với gói thời gian Tsi ở thời điểm bắt đầu của QT và gói thời gian Tvi ở thời điểm của pha hợp thức. Mọi đối tượng Oj ghi thời điểm đọc cuối cùng và thao tác ghi như TDJ và WRj được gọi là số hiệu phiên của Oj. Tập hợp dữ liệu được đọc bởi QT t1 trong pha thực hiện được thiết lập bởi Ri. Dữ liệu được thiết lập do ghi và Wi. Chuỗi các QT là TV của pha hợp thức.

Pha thực hiện: Bắt đầu TM khi nhận được tín hiệu bắt đầu từ máy Khách. Một không gian được tạo ra cho mỗi QT. Chúng ta giả thiết rằng được quản lí bởi TM hơn là phân tán ra các vị trí khác. Phiên bản của dữ liệu với số hiệu của nó được đọc vào vùng không gian đó nơi diễn ra thao tác cập nhật tương tự như đối với dùng phiên của file. Việc ngừng được đưa ra bằng việc xoá bỏ đi QT cũng như vùng không gian. Vì xuất hiện dấu hiệu yêu cầu được uỷ thác nên QT kết thúc. Và chuyển sang pha hợp nhất.

Pha hợp nhất: Các QT phải được nhất quán giữa QT được yêu cầu với những QT được phân tán khác đang được thực hiện tại những vị trí TPS tạo tành chuỗi. Để hợp nhất hiệu quả Tm sử dụng giao thức hợp nhất 2 pha với nó như là sự liên kết. Các thông số Ri, WI và Tvi cho QT t1 để yêu cầu TM hợp nhất. QT bao gồm các bước sau.

-Tính nhất quán của Ti được phản hồi nếu Tvi<TVk tất cả đều phải thực hiện liên tục với TV.

- Ti được chấp nhận nếu nó không vượt quá bất kì Tk nào. Ti liên quan đến Tk.

- Pha thực hiện của Ti phủ lên pha cập nhật của Tk và Tk hoàn thành pha cập nhật trước Tvi. Sự hợp thức của Ti được chấp nhận nếu Ri giao với Wk=rỗng.

- Pha thực hiện của Ti phủ lên sự hợp nhất và pha cập nhật của Tk và hoàn thành sự thực thi của nó trước Tsi. Sự hợp nhất của Ti được chấp nhận nếu Ri giao Wi=rỗng và Wi giao Wk=rỗng.

Các trường hợp trên được minh hoa ở hình 6.8. Vì Tvi phải >TVk và Tk phải xong trước Tk. Nếu cả 2 đều không thoả mãn những xung đột chỉ được kiểm tra đồng bộ với Wk của Tk. Trường hợp 3 nói rằng Tk sẽ hoàn thành. Nếu giao của thao tác đọc là Ti và ghi là Tk không xảy ra Ti có thể đã đọc một số dữ liệu đang được cập nhật bởi Tk. Xung đột phải xảy ra trong trường hợp 4 vì Tk có thể đã thành công. Thêm vào đó giao

của thao tác ghi Ti và Tk phải không có. Vì vậy thời điểm cập nhật của Tk có thể ảnh hưởng tới thời điểm cập nhật của Ti. Giai đoạn hợp nhất phải được sẵn sàng cho sự uỷ thác của bộ lưu trữ.

Hợp nhất liên quan đến sự tính toán. Đối với sự thực hiện đơn giản hơn ta giả sử rằng mỗi quá trình chỉ có một sự hợp nhất. Nếu có đòi hỏi có sự quản lí trong tiến hành hợp nhất đáp lại là trạng thái bận tới TM. Đòi hỏi của Tm không thực hiện nữa đã và sẽ tiếp tục lại. Sẽ xuất hiện số hiệu cho đối tượng dữ liệuu chia xẻ nhưng ở xa. Nó được so sánh với Tvi. Với WR phải nhỏ hơn Tvi của ti. Các QT phải đợi đến pha cập nhật cùng với hợp nhất của mỗi thời điểm Tm và khoảng không gian là thích hợp.

Pha cập nhật: Pha hợp nhất đối với bài toán điều khiển đồng bộ tối ưu tương tự như thời điểm ghi lại của phương thức gói thời gian. Pha cập nhật cũng tương tự như pha uỷ thác nhưng có một số điểm khác nhau chính sau. Thời điểm ghi thăm dò có thể dừng trong khi sự hợp nhất không thể không diễn ra. Cập nhật phải được uỷ thác trong thời gian TV cho QT của nó. Hoặc là không đáng được hoặc là không cần thiết để hoàn thành QT hợp thức cho đến khi xong giai đoạn cập nhật. Vì QT đã xong và việc quản lí khong gian nhớ có thể thực hiện được để những phiên bản dùng cho về sau. Vấn đề có thể được giảm bớt giả thiết rằng quản lí việc cập nhật có thể được tiến hành liên tục theo chuỗi bởi TM.

Giao thức điều khiển đồng bộ tối ưu đòi hỏi qá hợp thức và uỷ thác cho mọi QT là phức tạp. Dường như không tính đơn giản của nó.

6.4. Nhân bản dữ liệu và file

Đối tượng dữ liệu và file thường được nhân bản trong những hệ thống mạnh và khả dụng. Với việc nhân bản bước tiến cao hơn đạt được bằng việc cho phép xảy ra truy cập và tính khả dụng cao có thể dạt được bằng xử lí được dư thừa dữ liệu. Có phương pháp song song và các QT được xem xét trong hệ thống phân tán. Tuy nhiên chúng không khả dụng trừ phi chúng cũng cung cấp sự nhân bản và tranh chấp. Sự nhân bản có nghĩa là máy Khách cũng không nhận thấy sự tồn tại của QT này. Điều kiện của tính trong suốt sự tranh chấp là việc chia xẻ dữ liệu giữa các máy Khách cần được tránh.

Hai vấn đề cần được chỉ dẫn. Trước hết thao tác trên những vị trí khác nhau yêu cầu phải rõ ràng có thể với tất cả hoặc không. Thuộc tính này có thể đạt được bằng việc sử dụng khoá hoặc giao thức uỷ thác hai pha. Vấn đề thứ hai là sự tranh chấp sẽ được xử lí bằng việc thực hiện theo chuỗi thao tác. Chúng ta đưa ra các phương thức khác nhau đối với bài toán xung đột trông các phần trước. Nếu dữ liệu được nhân bản lại chúng ta cần có thêm yêu cầu thứ 3. Việc cập nhật cũng là quan trọng. Cứ cho rằng khả năng quản lí dữ liệu sẽ đảm bảo cho quá trình nhân bản dữ liệu được tốt chẳng hạn máy Khách nhận được phiên bản đơn của dữ liệu và sự thực hiện của QT đối với dữ liệu được nhân bản kết quả phải như thực hiện trên đối tượng dữ liệu không nhân bản. Các QT trên kết hợp lại gọi là sự thực hiện chuỗi phiên bản được nhân bản đối với hệ phân tán.

Cập nhật không chỉ giới hạn đối với những dịch vụ trong suốt. Nó được xem như chức năng chính đối với việc nhân bản dữ liệu trong bất kì hệ thống nào. Hiển nhiên dữ liệu trong hệ thống file có thể được định danh cho nhiều mục đích khác nhau. Đối với khả năng ứng dụng và tính khả dụng còn quan trọng hơn dữ liệu. Vì vậy yêu cầu đối với sự nhân bản dữ liệu thường ít nghiêm ngặt hơn tính trong suốt. Chúng ta sẽ thảo luận sự cập nhật và một số thay đổi hiệu quả với vấn đề quản lí.

Dựa trên 3 vấn đề: đa tán phát, giao dịch cơ bản và cập nhật cơ bản

Đa tán phát nguyên tử (Atomic multicast): các thông điệp với các dạng khác nhau được phân tán tới tất cả các nhóm khác nhau theo thứ tự phân tán và tuân theo thứ tự tổng thể.

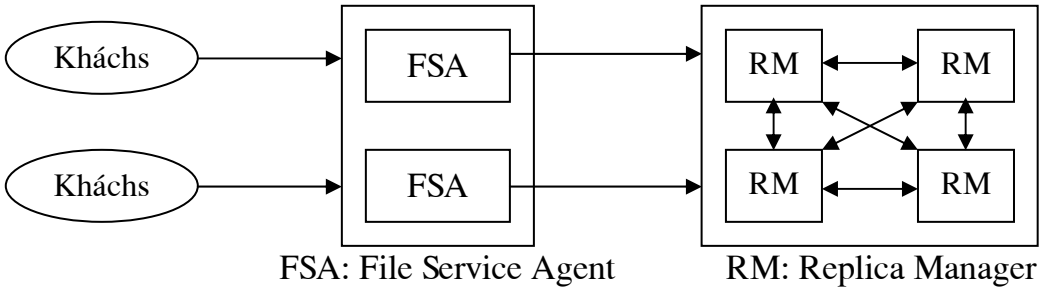
Giao dịch nguyên tử (Atomic transaction): Thao tác đối với mọi QT có thể xảy ra tất cả hoặc là không và các thao tháo giữa nơi tranh chấp được thực hiện theo thứ tự giống nhau.

Cập nhật nguyên tử (Atomic update): Cập nhật tới các bản sao và theo chuỗi.

Đối với các thông điệp khác nhau liên quan tới thứ tự phân tán dựa trên các giao thức như TCP. Phụ thuộc vào các yêu cầu khác nhau mà thứ tự có thể là FIFO. Tính đa dạng là một nhân tố đặc biệt mà các thông điệp với thao tác này có thể ảnh hưởng tới mọi cái khác. Tương tự chúng ta cũng chú ý rằng cập nhật cũng xem như là một QT mà mọi QT cập nhật có thể có tranh chấp. Đối với vấn đề quản lí nhân bản có thể không cần chặt chẽ đối với một số ứng dụng. Chẳng hạn những sai sót khi nhân bản có thể được phép trong thời gian một bản sao đang được truy cập bởi Khách dòng và Khách dòng có thể quan tâm đến phiên bản dữ liệu có thể là theo các nhóm hoặc theo quan điểm thích ứng. Chúng ta sẽ đề cập một cách cụ thể hơn.

6.4.1. Kiến trúc quản lí nhân bản

Kiến trúc hệ thống chung trên hình 6.9. đối với việc quản lý các dữ liệu được nhân bản.



Hình 6.9. Một kiến trúc đối với bản sao

Khách có thể chọn một hoặc nhiều hơn Tác nhân dịch vụ file (File Service Agent: FSA) đối với vấn đề truy cập dữ liệu. Dịch vụ file xem như là các bộ dịch vụ nối với bộ quản lí nhân bản (RM) để cung cấp sự dễ dàng nhân bản của Khách. Một FSA có thể liên lạc với một hoặc nhiều RM cho việc đọc hoặc cập nhật dữ liệu. Tuỳ thuộc vào giao thức điều khiển đã được thực hiện như nào mà phản hồi của việc điều khiển nhân bản có thể được chia giữa FSA và RM. Cấu trúc là mô hình clent/serverr khác biệt với cấu trúc ở hình 6.6. Nếu mỗi bộ điều khiển nhân bản được xen với một dịch vụ file chúng ta có ứng dụng đặc biệt nhóm những Khách xử lí đối với vấn đề riêng biệt cho việc cùng soạn thảo một vấn đề nhưng hoàn toàn đó là đối tượng dữ liệu được nhân bản. Thao tác xử lí dữ liệu có thể là đọc hay cập nhật. Thao tác cập nhật có thể được định nghĩa cụ thể hơn là ghi hoặc ghi cho việc sửa đọc. Dưới đây sẽ dùng cách thức ghi và cập nhật. Từ những đòi hỏi của người dùng thao tác đọc cần được địa chỉ hoá duy nhất cho việc nhân bản. Tuy nhiên nhân bản là dễ nhận thấy đối với người dùng và dịch vụ file được gọi ra bởi Khách có thể được yêu cầu bởi giao thức quản lí nhân bản tác động một một hoặc nhiều hơn một bộ nhân bản để đảm bảo cho việc đọc dữ liệu là sớm nhất. Có 3 vấn đề đối thao tác đọc:

Đọc nguyên thủy (Read-one-primary): FSA chỉ có thể đọc từ RM nguồn để đảm bảo tính chính xác.

Đọc tiêu biểu (Read-quorum): FSA phải đọc từ một số RM đại diện để xác định dữ liệu hiện tại

Tương tự như những điểm nổi bật của hệ thống thao tác ghi sẽ được địa chỉ hoá tới tất cả bản sao. Tuy nhiên những ứng dụng không cần thiết để chắc chắn như những yêu cầu cập nhật, có thể được được gửi tới một số hoặc tất cả các bản sao trong một thời điểm nào đó. Có một số vấn đề sau:

Ghi toàn bộ (Write-all): Cập nhật tất cả RMs bởi thao tác ghi là cần thiết. Thao tác ghi tuần tự phải đợi cho đến khi thao tác ghi trước kết thúc và được uỷ thác.

Ghi toàn bộ theo khả năng (Write-all-avaiable): Việc cập nhật tới những RM sẵn có. Với việc xảy ra lỗi bộ quản lí nhân bản lưu trạng thái của nó tới trạng thái hiện tại trước khi thực hiện việc truy cập tới các bản sao.

Ghi tiêu biểu (Write-quorum): Việc cập nhật được thực hiện với một số bộ quản lí đại diện

Ghi ngẫu hứng (Write- gossip): Việc cập nhật thực hiện trực tiếp tới bất kì RM nào và được gửi tới những RM rỗi kề cận.

Sau đây là mô tả về sự kết hợp thao tác đọc và ghi bản sao

6.4.2. Chuỗi một bảo sao

Nếu cả thao tác đọc và ghi là trực tiếp tới bộ quản lí nhân bản thì sự nhân bản là không cần thiết. Tất cả những thao tác phải thực hiện liên tiếp bởi RM gốc. Các RM sau chỉ cần thiết trong trường hợp bản gốc bị lỗi. Tính nhất quán là dễ dàng đạt được nhưng xung đột không được xem xét đến. Để tạo ra được sự đồng bộ thao tác đọc sẽ được thực hiện tại bất kì RM nào. Tuy nhiên mục này chỉ xét đến tính nhất quán vì việc cập nhật từ RM ban đầu tới RM sau là không liên tục. Chúng ta có thể được cho chép một thao tác cập nhật phải được chú ý đến. Nếu như giao thức cho việc này đã có chúng ta tiến hành từ tại bất kì RM nào đáp ứng yêu cầu của Khách. Đây là giai đoạn đọc-một/ghi-toàn bộ. Tính nhất quán trong nhân bản dữ liệu có thể dùng giao thức điều khiển đồng bộ chuẩn như khoá 2 pha hoặc giao thức gói thời gian. Nếu thao tác đọc ghi là không dễ dàng giao thức đọc-một/ghi-toàn bộ dẫn đến sự thực liên tiếp từng bản ghi một, sự thực hiện không phải cùng một đối tượng. Phương thức write-all không thực tế đối với quản lí nhân bản. Đối tượng dữ liệu được nhân bản dẫn đến sự cố. Vì vậy việc cập nhật chỉ thực hiện trên các bản sao. Thao tác đọc-một/ghi-toàn bộ sẵn có đường như thích hợp hơn. Tuy nhiên phương thức này sẽ làm cho thao tác liên tục với bản sao trở nên phức tạp hơn. Dưới đây là một ví dụ minh hoạ cho vấn đề này.

t0:b(t) W(x) W(y) et

t1: b(t) R(x) W(y) et

t2: b(t) R(y) W(x) et

To bắt đầu với x và y với thao tác ghi (W). Nó cho phép 2 QT khác xảy ra đồng thời t1 và t2. Mỗi thao tác đọc đ và thao tác ghi cùng chia xẻ đối tượng x và y. Sự thực hiện đúng khi t1 và t2 là thực hiện liên tiếp. Đó là hoặc t1 đọc x được ghi bởi t2 hoặc t2 đọc y được ghi bởi t1. Nó có thể được xem xét như không có sự thực hiện liên tiếp đối với 2 QT trừ phi danh mục là thực hiện liên tiếp. Bây giờ giả sử rằng chúng ta sao x tới x và xb và y tới yc và yd. Và lỗi xuất hiện tới xc và

tc. Hai quá trình trở thành:

t1: bt R(xc) (yd fails) W(yc) et

t2: bt R (yd) (xc fails) W(xb) et

Sau khi sự cố của xc và yd sao tới X và Y và chỉ có Xc và Yc. Nếu chúng ta sử dụng đọc-một/ghi-toàn bộ có sẵn t1 và t2 là liên tiếp vì không ảnh hưởng từ thao tác đọc và ghi. Có sự không thích ứng với hệ thống một bản sao bởi vì cả t1 và t2 không biết đối

tượng được ghi bởi thao tác khác. Đó là sự thực hiện không phải theo cách thức một bản sao. Thêm vào đó tham chiếu cần phải được thêm vào để đảm bảo tính chất một bản sao. Từ ví dụ trên chúng ta thấy rằng sự cố và phục hồi phải được thực hiện liên tiếp. Đối với t2 để đọc Yd nếu Yd lỗi trước khi t2 hoàn thành. Lỗi của Yd trong t1 làm cho t2 phải dừng và lại quay lại thao tác đọc trên Yd. Thêm vào đó lỗi có thể xuất hiện trước khi bắt đầu QT. KHi t2 bắt đầu nó sẽ phải liên hệ với Yc vì Yd đã có lỗi rồi. Ảnh hưởng giữa W(Yc) trong t1 và R(Yc) trong t2 là có thể.

Chúng ta xem xét ví dụ trên với ảnh hưởng giữa t1 và t2 cho việc truy cập Y là không rõ ràng bởi vì đã có sẵn bản sao cho W(Y) trong t1 mà không bao giờ gôm Yd được đọc bởi t2. Chúng ta cũng giải quyết vấn đề t2 yêu cầu được đọc Yc và Yd khi truy cập Y. Điều này có thể được thực hiện nếu việc đọc sẽ tạo ra bản sao của QT cùng với việc ghi của QT khác. Điều này tương ứng với phương thức đọc-tiêu biểu được thảo luận về sau.

6.4.3. Hình thức cử đai diên

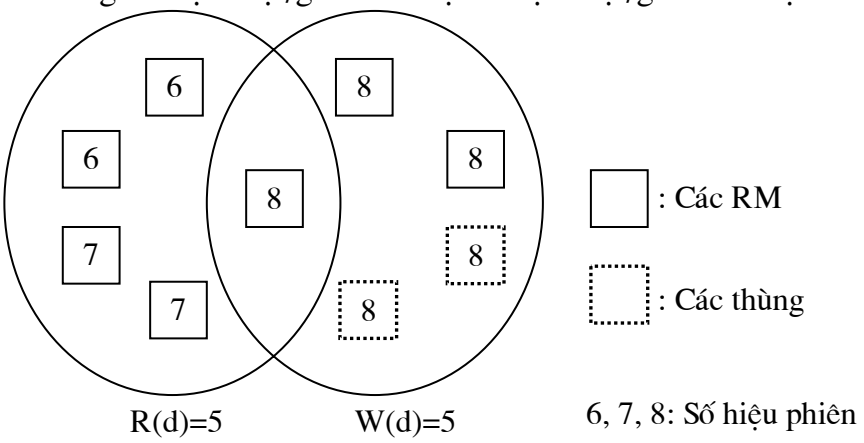
Hình thức này yêu cầu mỗi thao tác đọc bản sao dữ liệu đối với một số đại diện của bộ quản lí nhân bản và với thao tác ghi cũng thế. Sử dụng 2 hình thức này ảnh hưởng giữa các thao tác truy cập đối với những dữ liệu được nhân bản có thể được đại diện bởi có sự phủ lên nhau. Xem xét V(d) như là tổng số phiên bản của dữ liệu d. Luật xen phủ của đọc và ghi là

Xung đột ghi - ghi: W(d)> V(d)

Xung đột đọc-ghi: R(d)+ W(d) > V(d)

Mỗi khi một xung đột biểu lộ, giao thức khoá 2 pha chuẩn hay tem thời gian có thể được dùng để thực hiện dãy một bản sao. Để đảm bảo rằng Khách đọc gần như toàn bộ dữ liệu số hiệu phiên bản có thể liên quan tới dữ liệu được nhân bản. Thao tác đọc tới bản sao R(d), giá rị với số hiệu cao nhất được cho. Thao tác ghi với W(d) và thêm 1 với số hiệu cao nhất được tìm thấy đây là số hiệu mới với mỗi lần đọc hoặc ghi.

Hầu hết các ứng dụng R(d) được chọn là nhỏ hơn W(d). Trong trường hợp đặc biệt R(d)=1 và W(d) =Vd). Nếu W(d)<V(d) lỗi có thể được cho phép nhưng R(d) phải lớn với hình thức ghi toàn bộ theo khả năng. Giao thức đọc-tiêu biểu/ghi-tiêu biểu là dung hoà tốt giữa đọc-một/ghi-toàn bộ và đọc-một/ghi-toàn bộ theo khả năng. Nếu R(d)<< W(d) thì đòi hỏi về bộ nhờ là lớn. Để giảm đòi hỏi này thao tác ghi có thể sử dụng một bộ đánh dấu. Bộ này không chứa toàn bộ về thông tin cần thiết như số hiệu và thông tin xác định dữ liệu tham dự trong thao tác ghi.

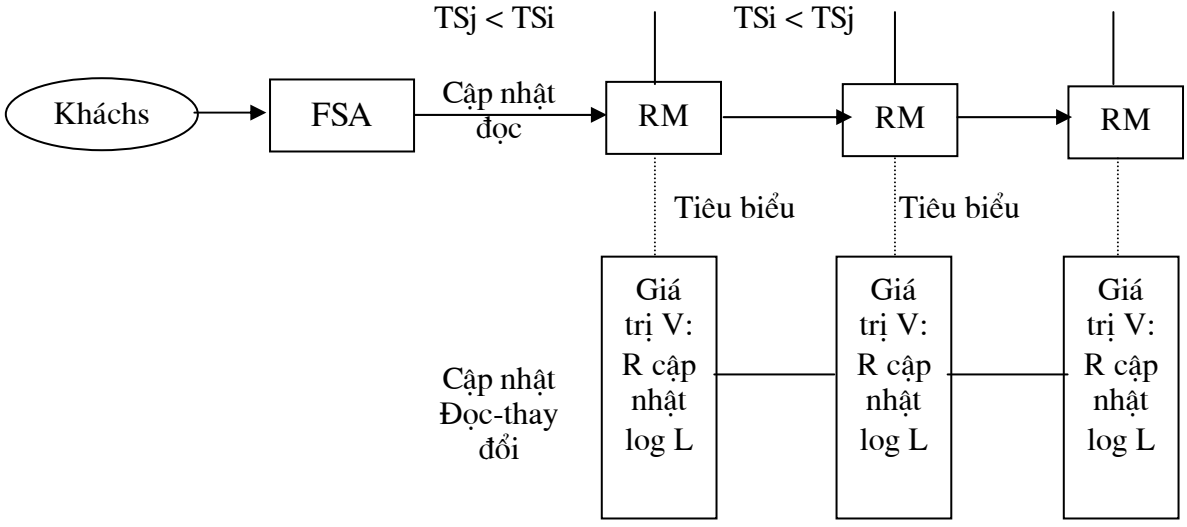


Hình 6.10. Một đọc-tiêu biểu/ghi-tiêu biểu với thùng

6.4.4. Nhân cập nhật ngẫu hứng

Nhiều ứng dụng của nhân bản dữ liệu không có tính nhất quán với việc thực hiện chuỗi các thao tác trong phương thức đọc-một ghi rất cả sẵn có hay là đối với vấn đề cập nhật

dữ liệu trong giao thức đọc đại diện. Nếu việc cập nhật là không tuần tự đọc và ghi thì nó có thể khó thực hiện người ta đưa ra một phương thức mới. Cách thức đọc một cách ngẫu hứng hơn được gọi là giao thức cập nhật theo ngẫu hứng. Trong giao thức này thao tác đọc và ghi là trực tiếp bởi dịch vụ file FSA và RM. FSA thực hiện nhân bản dữ liệu từ Khách còn RM cập nhật dữ liệu bằng việc thay đổi thông tin giữa các bộ dịch vụ. Nói chung mục đích của phương thức này là đem khả năng sẵn có cho việc nhân bản và cập nhật dữ liệu.



Hình 6.11. Một kiến trúc ngẫu hướng

Xem xét theo minh hoạ ở hình 6.11. Đầu tiên là thao tác cập nhật dữ liệu, sử dụng phương thức đọc sửa trên đối tượng dữ liệu. Có thể thấy như trên hình vẽ, RM được liên kết với TS.

Giao thức ngẫu hứng cơ sở

Đọc: TSf của FSA được so sánh với Tsi của RM. Nếu TS<RM thì RM sẽ lưu dữ liệu và FSA sẽ nhận dữ liệu đó. Ngược lại FSA sẽ đợi cho đến khi RM sẽ cập nhật dữ liệu và lưu giữ nó. Thay vì thời gian FSA có thể liên hệ với RM khác.

Cập nhật: FSA với TSf. Nếu TSf>Tsi thì việc cập nhật sẽ được thực hiện, nếu không chờ cho sự cập nhật mới. RM có thể có cách thức riêng thuận tiện bằng cách sử dụng giao thức tự do. Nếu TSf<=Tsi thì sự cập nhật đã được thực hiện hoặc là đang được xử lí. Phụ thuộc vào các ứng dụng mà bỏ qua thao tác ghi chỉ có thao tác đọc và cập nhật.

Ngẫu hứng: Thông báo ngẫu hứng sẽ chuyển việc nhân bản từ bộ RM i sang bộ RM j sẽ đươc thực hiện nếu TSj >Tsi.

Ta có thể sử dụng vectơ gói thời gian thay cho những gói thời gian đơn lẻ. Cách thức này là một bước tiến, vectơ gói thời gian bao gồm số hiệu được sắp xếp (hay số hiệu gói) đại diện cho số thứ tự cập nhật tại RMi1, RM2, RM3. Thao tác ghi với TSf=<2,2,2> bằng hoặc ít hoặc ít hơn Tsi thi được phép thực hiện. TSf=<2,5,5> cần phải đợi nhưng chúng ta có nahạn thấy sự thực hiện sẽ diễn ra tại RM2 và RM3.

Chúng ta đã sử dụng khái niệm tem thời gian, việc duy trì thứ tự là cần thiết nếu thao tác cập nhật là sửa đọc. Kết quả của dữ liệu được tăng lên 2 lần từ sự khác biệt khi thứ tự được duy trì hay thao tác bị bỏ qua.

Như hình 6.11 và 6.12 mỗi quản trị nhân bản được liên kết với 2 vectơ tem thời gian V và R và L.Vchứa tem thời gian mang giá trị hiện tại của dữ liệu. R cho biết những

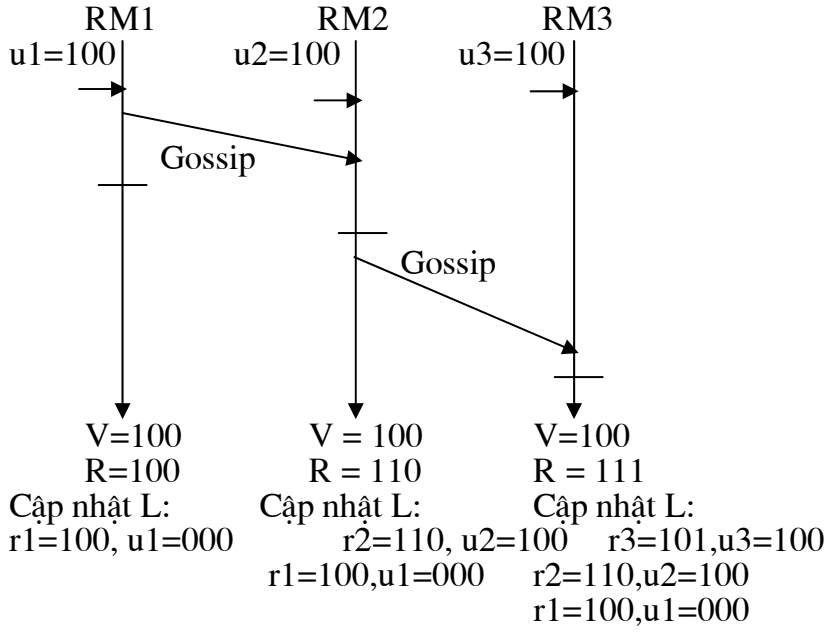
nhóm yêu cầu về cập nhật. Chẳng hạn R=<2,3,4> tại RM cho thấy 2 yêu cầu thao tác cập nhật từ và 3 và 4 thao tác cập nhật tại RM3 và RM4.

Giao thức ngẫu hứng có thứ tự

Đọc: FSA với TSf chứa yêu cầu q. Nếu q<=V thì RM sẽ mang giá trị tới FSA. Nếu không FSA sẽ đợi thực hiện.

Cập nhật: FSA với TSf với thao tác cập nhật u. Nó được so sánh với V. Nếu u<V thì coi như thao tác sẽ chậm và trở lại RM bởi vì thứ tự thực hiện là cần thiết. FSA sẽ không liên hệ nữa, việc đọc sẽ chuyển cho thao tác cập nhật. Trong trường hợp thông thường u>=V thì ta cũng có thể suy ra u=V. Như vậy trong bất kì trường hợp thì việc cập nhật cũng được chấp nhận. Bộ quản lí nhân bản RM tiếp nhận thêm thành phần i của R bằng việc chỉ ra một hoặc nhiều cập nhật đã được thao tác cập nhật u trong trường hợp đăng khí vào L. Nếu u=V thì việc cập nhật được thực hiện, tem thời gian V được trộn lẫn với r. Kết quả của thao tác đều xảy ra trước khi được uỷ thác. Nếu nhiều hơn một trường thoả mãn điều kiện này thì các quá trình của nó phải đăng kí theo trật tự của nó. Thao tác cập nhật khác chờ đợi để đăng kí vào tình trạng ổn định. Nó bị thay đổi khi RM có thao tác cập nhật hoá tự do tác động. Việc cập nhật khi hoàn thành sẽ được đánh dấu để không lặp lại, quá trình cập nhật không xoá luôn vì nó cần định danh ra Rms mới.

Ngẫu hứng: Thông điệp từ RM_j tới RM_i mang vectơ tem thời gian R_j của RM_j và đăng kí vào L_j. R_j kết hợp với vectơ tem thời gian R_i của RM_i. Việc đăng kí vào RM_i được thực hiện với L_j trừ việc cập nhật các trường có $r<=V_i$. Các trường này phải được khai báo bởi RM_i. Tập hợp đăng kí được xem như là tự do nếu nó phải chứa đầy đủ thông tin về RM và nhiều RM khác.



Hình 6.12. Kịch bản ngẫu hứng thứ tự nguyên nhân

Hình 6.12. minh hoạt 3 thao tác của quá trình cập nhật và đăng kí theo trật tự. Tất cả mọi vectơ tem thời gian được khởi tạo giá trị <0,0,0>. U1=<0,0,0> định hướng trực tiếp tới RM1 được thực hiện u1>=V1 khi đó R1=<1,0,0>. Bản ghi cập nhật của r1 và u1 được tập trung trong L1 và được thực hiện khi đó V1=<1,0,0>. Trong khi đó L1 chỉ chứa quá trình cập nhập tự do tới RM2, QT cập nhật U2, U3 từ RM2, RM3 một cách riêng biệt. Chúng phải đọc dữ liệu từ RM1 trước vì có tem thời gian là <1,0,0> U2 xảy ra tại RM2. Vì tem thời gian lớn hơn V2=<0,1,0> nên nó được định vị trong L2 sau khi R2 được cập nhật tới <0,0,0> được đặt trong L3. Ví dụ trên chỉ ra những khái niệm và thao tác cơ bản để cập nhật ngẫu hứng.

BẢNG GIẢI NGHĨA CỤM TỪ GHI TẮT VÀ THUẬT NGỮ

Cụm	*Tiếng Việt*	*Tiếng Anht*	*Giải nghĩa*	*Trg*
HĐH/ OS	Hệ điều hành	Operating System	Bộ phần mềm	6
	Đa chương trình	multiprogramming	Hoạt động HĐH	11
	Phân chia thời gian	time shared	Hoạt động HĐH	11
	Đa người dùng	multi-users	Hoạt động HĐH	12
	Trạm cuối	terminal	Thiết bị	12
	Lượng tử thời gian	time quantum	Hoạt động HĐH: lập lịch	12
	Hệ điều hành kết hợp	Combination Operating System		13
	Siêu máy tính	supercomputer	Kiểu máy tính	13
CTĐ	Chuyển thông điệp	message passing	Công việc - loại dịch vụ	18
ĐBQT	Đồng bộ quá trình	process syncronization	Yêu cầu điều khiển quá trình	15
TTLQT	Truyền thông liên quá trình	interprocessing comunication	Tương tác các quá trình	15
	(Bộ-) lập lịch	schedule (-er)	(Môđun) lập lịch	17
	Bộ giám sát	monitor	Kiểu dữ liệu đồng bộ	18
	Điểm hẹn	rendezvous (Ada)	Kiểu dữ liệu đồng bộ	18
DOS	HĐH phân tán	Distributed OS	Kiểu HĐH	8
NOS	HĐH mạng	Network OS	Kiểu HĐH	8
CAS	Hệ tự trị cộng tác	Cooperative Autonomous System	Kiểu HĐH	8
SPOOLING		Simultaneous Peripheral Operation OnLine	Vào - ra của HĐH với đĩa từ nên vào - ra nhanh	10
MFT		Multiprogramming with Fixed number of Tasks		11
MVT		Multiprogramming with Variable number of Tasks		12
TSS		Time Shared System		12
VLSI	Vi mạch tích hợp rất lớn	Very Large Scale In.		13
SISD	Đơn lệnh - đơn dữ liệu	Single Data Single Instruction		14

SIMD	Đơn lệnh - đa dữ liệu	Single Instruction Multiple Data		14
MISD	Đa lệnh - đơn dữ liệu	Multiple Instruction Single Data		14
MIMD	Đa lệnh - đa dữ liệu	Multiple Instruction Multiple Data		14
API	Giao diện trình ứng dụng	Application Program Interface		16
HAL	Mức trừu tượng phần cứng	Hardware Abstraction Layer		16
SPI	Giao diện cung cấp dịch vụ	Service Provider Interface		16
PE	biểu thức đường đi	Path Expression		18
CR	Khoảng tới hạn	Critical Region		18
CCR	khoảng tới hạn có điều kiện	Condition Critical Region		18
CSP	Bộ các quá trình tuần tự truyền thông	Communicating Sequential Processes		18
	giao diện	interface		
	bế tắc	deadlock	Hiện tượng tập quá trình không thực hiện	19
	điều khiển trang	paging	Một kiểu điều khiển bộ nhớ gián đoạn	19
	điều khiển segment	segmentation	Một kiểu điều khiển bộ nhớ gián đoạn	19
	Trạm làm việc	workstation		20
	liên thao tác	interoperability		21
	hỗn tạp	heterogenous		7
	giao vận	transport service		
	điểm - điểm	peer to peer		21
OSI				21
ISO				21
FS	Hệ thống File	File system		21
NFS	Hệ thống file mạng	Network FS		21
		socket		21
RPC	lời gọi thủ tục từ xa	Remote Procedure Call		21
	đăng nhập từ xa	remote login		21

	chuyển file	file transfer		21
	duyệt mạng	network browsing		21
	thực hiện từ xa	remote execution		21
	không đồng bộ	asynchronous	dị bộ	22
	Giao thức truyền mail đơn giản	Simple Mail Transfer Protocol: SMTP		22
URL	Bộ định vị tài nguyên thống nhất	Universal Resource Locator		22
HTML	Ngôn ngữ đánh dấu văn bản	HyperText Markup Language		22
MIME		Multipurpose Internet Mail Extension		23
	tiểu dụng	applet		23
	trong suốt	transparency		24
	trong suốt đồng thời	concurrency transparency		24
	trong suốt truy nhập	access transparency		34
	trong suốt định vị	location transparency		34
	trong suốt di trú	migration transparency		34
	trong suốt nhân bản	Replication transparency		34
	trong suốt song song	parallelsm transparency		34
	trong suốt lỗi	failure transparency		34
	trong suốt hiệu năng	performance transparency		34
	trong suốt kích thước	size transparency		34
	trong suốt duyệt lại	revision transparency		34
CSCW	Làm việc cộng tác được hỗ trợ bằng máy tính	Computer Supported Cooperative Work		26
ODP	Quá trình phân tán mở	Open Distributed Processing		26
CORBA	Kiến trúc môi giới yêu cầu đối tượng chung	Common Object Request Broker Architeturre		26

	phần mềm lớp giữa	middleware		26
	thứ lỗi	fault tolerance		27
	tính nhất quán	consistency		28
	tán phát	broadcast		
	tán phát tin cậy	reliable broadcast		
	giao dịch	transaction		22
	thuộc- người dùng, liên người dùng, và liên nút	intrauser, interuser, internode		
	dịch vụ nguyên thủy đồng bộ	syschronous primitive		38
	dịch vụ kết khối	blocking primitive		38
	dịch vụ	service	dịch vụ của hệ thống	
	phục vụ	server	đ/tượng cung cấp dịch vụ	
	công tác	coordination		
PAD	thiết kế và giải thiết kế gói	packet assembling and deassembling		39
	xâu bộ xử lý	processors - pool		39
	trạm làm việc-phục vụ	workstation - phục vụ		39
	cổng	port		48
	tán phát bội	multicast		50
	quảng bá	broadcast		50
	bộ nhớ chia xẻ phân tán	distributed shared memory		51
DCE	Môi trường tính toán phân tán	Distributed Computing Enviroment		51
	hàng đợi kết nối với QT ngắn nhất	join-to-the-shortest queue		140
	Thực hiện từ xa	Remoce execution		145
	Dịch vụ từ xa	remote service		145
	TĐ thông dịch	intepretive message		145
	Lệnh gọi từ xa	remote command		
	giao thức truyền thông ứng dụng	Application communication protocol		146
	Phiên bản	version		9