LỜI CẢM ƠN

Trong suốt khóa học 2005 – 2009 tại trường Đại Học Dân Lập Hải Phòng với sự giúp đỡ của quý thầy cô và giáo viên hướng dẫn về mọi mặt, từ nhiều phía nhất là trong thời gian thực hiện đề tài, nên đề tài của em đã được hoàn thành đúng thời gian quy định.

Em xin gửi lời cảm ơn chân thành nhất tới thầy giáo hướng dẫn Th.s Nguyễn Trịnh Đông đã tận tình hướng dẫn, giúp đỡ, tạo điều kiện để em hoàn thành khóa luận này.

Em xin gửi lời cảm ơn chân thành tới Bộ môn Công Nghệ Thông Tin cùng toàn thể các thầy cô trong khoa cũng như toàn thể các thầy cô trong Trường đã giảng dạy những kiến thức chuyên môn làm cơ sở để em thực hiện tốt cuốn luận văn tốt nghiệp này và đã tạo điều kiện thuận lợi để em hoàn thành khóa học.

Em xin chân thành cảm ơn !

Hải Phòng, ngày 28 tháng 6 năm 2009

Sinh Viên

Đặng Thị Hải

MỤC LỤC

LỜI GIỚI THIỆU

Cơ sở dữ liệu phân tán là mô hình lưu trữ dữ liệu rất quan trọng trong các hệ thống thông tin lớn và ngày càng phát triển. Hiện nay, CSDL phân tán được ứng dụng trong hầu hết các hệ thống thông tin trong các lĩnh vực như ngân hàng, thương mại, giáo dục, doanh nghiệp ….

Đặc trưng chính của CSDL phân tán là có rất nhiều các thao tác truy cập tới một hoặc nhiều vị trí khác nhau trên mạng để trao đổi dữ liệu. Do vậy, vấn đề là xảy ra tương tranh trong quá trình trao đổi thông tin. Trong hệ cơ sở dữ liệu phân tán việc điều khiển tương tranh là bài toán rất quan trọng.

Trong đồ án tốt nghiệp này em nghiên cứu và tìm hiểu nội dung ***"Các phương pháp điều khiển tương tranh và truy cập dữ liệu trong cơ sở dữ liệu phân tán".***

Nhằm hiểu rõ vấn đề tương tranh, cách thức điều khiển tương tranh và truy cập dữ liệu trong cơ sở dữ liệu phân tán để đảm bảo sự nhất quán của dữ liệu khi có các thao tác tác động lên cơ sở dữ liệu.

Đồ án được chia thành 3 chương:

Chương 1: Tìm hiểu một số đặc điểm của cơ sở dữ liệu phân tán.

Chương 2: Giới thiệu về các thao tác truy cập đến cơ sở dữ liệu phân tán.

Chương 3: Timg hiểu các phương pháp điều khiển tương tranh và truy cập dữ liệu trong cơ sở dữ liệu phân tán.

CHƯƠNG 1: GIỚI THIỆU CƠ SỞ DỮ LIỆU PHÂN TÁN

1.1 CƠ SỞ DỮ LIỆU.

1.1.1 Định nghĩa cơ sở dữ liệu

Cơ sở dữ liệu là tập hợp các dữ liệu có liên quan với nhau, được lưu trữ trên máy tính, có nhiều người sử dụng và được tổ chức theo một mô hình. Dữ liệu là những sự kiện có thể ghi lại được và có ý nghĩa.

1.1.2 Các tính chất của cơ sở dữ liệu

- Một cơ sở dữ liệu biểu thị một khía cạnh nào đó của thế giới thực. Những thay đổi của thế giới thực phải được phản ánh trung thực vào cơ sở dữ liệu.
- Một cơ sở dữ liệu là một tập hợp dữ liệu liên kết với nhau một cách logic và mang một ý nghĩa cố hữu nào đó.
- Một cơ sở dữ liệu được thiết kế và được phổ biến cho một mục đích riêng. Nó có một nhóm người sử dụng có chủ định và có một số ứng dụng được xác định phù hợp với mối quan tâm của người sử dụng.

1.1.3 Hệ quản trị cơ sở dữ liệu.

Một hệ quản trị cơ sở dữ liệu là một tập hợp chương trình giúp cho người sử dụng tạo ra, duy trì và khai thác một cơ sở dữ liệu. nó là một hệ thống phần mềm phổ dụng, làm dễ quá trình định nghĩa, xây dựng và thao tác cơ sở dữ liệu cho các ứng dụng khác nhau.

1.2 CƠ SỞ DỮ LIỆU PHÂN TÁN.

Hệ cơ sở dữ liệu phân tán được phát triển dựa trên cơ sở dữ liệu và mạng máy tính. Cơ sở dữ liệu phân tán gồm nhiều cơ sở dữ liệu tích hợp lại với nhau thông qua mạng máy tính để trao đổi dữ liệu, thông tin… Cơ sở dữ liệu được tổ chức và lưu trữ ở những vị trí khác nhau trong mạng máy tính và chương trình ứng dụng truy cập vào dữ liệu ở những điểm khác nhau đó.

1.2.1 Khái niệm cơ sở dữ liệu phân tán.

Vì yêu cầu của công ty, doanh nghiệp, đơn vị kinh doanh… về vấn để tổ chức kinh doanh sao cho kinh doanh có hiệu quả nhất và nắm bắt thông tin nhanh nhất khi các cơ sở của công ty ở những điểm xa nhau cho nên xây dựng một hệ thống làm việc trên cơ sở dữ liệu phân tán là phù hợp với xu hướng hiện nay.

Cơ sở dữ liệu phân tán nhằm mục đích đáp ứng cho việc lưu trữ và xử lý dữ liệu cho các tổ chức, công ty trong thời đại hiện nay đó là dữ liệu cần phải được cập nhật và lưu trữ tại nhiều vị trí địa lý khác nhau.

Cơ sở dữ liệu phân tán là tập hợp dữ liệu logic thuộc về cùng một hệ thống nhưng trải rộng ra nhiều điểm trên mạng máy tính. Như vậy có 2 vấn đề của cơ sở dữ liệu phân tán với tầm quan trọng tương đương nhau:

- Việc phân tán: Trong thực tế dữ liệu không đặt trên cùng một vị trí vì vậy đây là cơ sở để phân biệt cơ sở dữ liệu phân tán với cơ sở dữ liệu tập trung và cơ sở dữ liệu đơn lẻ.

- Liên quan logic: mặc dù được lưu trữ tại nhiều vị trí khác nhau nhưng có quan hệ với nhau, và có thể truy xuất tại mỗi vị trí theo giao diện chung.

1.2.2 Ưu nhược điểm của hệ quản trị cơ sở dữ liệu phân tán.

1.2.2.1 Ưu điểm.

Có nhiều nguyên nhân dẫn đến sử dụng cơ sở dữ liệu phân tán nhưng về cơ bản cơ sở dữ liệu phân tán có những ưu điểm sau:

- Lợi điểm về tổ chức và tính kinh tế: Tổ chức phân tán nhiều chi nhánh và dùng cơ sở dữ liệu phân tán phù hợp với các tổ chức kiểu này.

- Tận dụng những cơ sở dữ liệu sẵn có: Hình thành cơ sở dữ liệu phân tán từ các cơ sở dữ liệu tập trung có sẵn ở địa phương.

- Thuận lợi cho nhu cầu phát triển: Xu hướng dùng cơ sở dữ liệu phân tán sẽ cung cấp khả năng phát triển thuận lợi hơn và giảm được xung đột về chức năng giữa các đơn vị đã tồn tại và giảm được xung đột giữa các chương trình ứng dụng khi truy cập đến cơ sở dữ liệu.

- Giảm chi phí truyền thông: Trong cơ sở dữ liệu phân tán chương trình ứng dụng đặt ở địa phương có thể giảm bớt được chi phí truyền thông khi thực hiện bằng các khai thác dữ liệu tại vị chỗ.

- Tăng số công việc thực hiện: Cơ sở dữ liệu phân tán có nhiều thuận lợi trong việc phân tán dữ liệu như tạo ra các trình ứng dụng phụ thuộc vào tiêu chuẩn mở rộng vị trí làm cho các nơi có thể hỗ trợ lẫn nhau. Do đó tránh được hiện tượng tác nghẽn cổ chai trong mạng truyền thông hoặc trong các dịch vụ thông thường của toàn bộ hệ thống.

1.2.2.2 Nhược điểm.

- Kinh ngiệm thiết kế và ứng dụng chưa nhiều còn tồn tại nhiều vấn đề cần giải quyết.

- Các vấn đề của cơ sở dữ liệu phân tán thì phức tạp hơn nhiều so với cơ sở dữ liệu tập trung, đặc biệt là vấn đề khi cập nhật dữ liệu cũng như xử lý khi gặp lỗi.

- Vấn đề truyền thông: Bảo đảm an toàn thông tin cũng như chọn cấu hình mạng cho phù hợp.

- Vấn đề đồng bộ dữ liệu.

- Vấn đề an toàn dữ liệu: Nếu không có cơ chế bảo vệ hợp lý thì có khả năng những dữ liệu không mong muốn vẫn được truy xuất ra ngoài.

1.2.3 Các mức phân tán.

1.2.3.1. Mức tập trung.

Toàn bộ cơ sở dữ liệu được đặt ở một nơi và tất cả các yêu cầu của người dùng đều được xử lý tại nơi lưu trữ CSDL.

Ưu điểm: Việc quản lý tương đối dễ dàng.

Nhược điểm: Máy lưu trữ cơ sở dữ liệu phải có cấu hình đủ mạnh.

1.2.3.2 Mức dữ liệu ở một nơi, xử lý ở nhiều nơi.

Giả sử một máy tính nào đó trong mạng phát ra yêu cầu về dữ liệu thì yêu cầu đó được gửi về Server. Sau đó dữ liệu này được chuyển từ Server đến máy tính đó để xử lý dữ liệu.

Ưu điểm: Đáp ứng dữ liệu nhanh, máy lưu trữ dữ liệu không cần phải có cấu hình mạnh.

Nhược điểm: Tăng lưu lượng trên đường truyền dữ liệu.

1.2.3.3 Mức dữ liệu ở nhiều nơi và xử lý ở nhiều nơi.

Chia cơ sở dữ liệu DB ban đầu thành các đoạn dữ liệu con (DB_1, DB_2, …, DB_i, …, DB_n) được lưu trữ ở các máy tính, tại các địa điểm khác nhau. Khi máy tính thứ i có yêu cầu về dữ liệu thì quá trình diễn ra như sau:

Tìm kiếm xem các dữ liệu đang được lưu trữ trên máy nào.

- Xử lý dữ liệu ngay tại nơi tìm được.
- Sau khi xử lý xong sẽ chuyển kết quả xử lý đó về máy thứ i có yêu cầu ban đầu.

Ưu điểm: Xử lý công việc nhanh.

Nhược điểm: Rất khó khăn cho việc quản trị dữ liệu.

1.2.4 Các đặc trưng trong suốt của cơ sở dữ liệu phân tán.

Khái niệm trong suốt: trong suốt là sự che giấu mọi hoạt động phức tạp bên trong của hệ thống cơ sở dữ liệu phân tán, làm cho người sử dụng có cảm giác như đang làm việc với cơ sở dữ liệu tập trung.

1.24.1. Trong suốt phân tán.

- Khái niệm: Trong suốt phân tán cho phép cơ sở dữ liệu phân tán được xử lý như một cơ sở dữ liệu tập trung. Người dùng không phải quan tâm đến:

+ Cơ sở dữ liệu đã được phân đoạn như thế nào.

+ Các đoạn dữ liệu được lưu trữ ở những nơi nào.

- Các mức trong suốt phân tán:

+ Trong suốt phân đoạn: Cơ sở dữ liệu ban đầu mặc dù đã được phân chia thành các đoạn dữ liệu con. Nhưng trong truy vấn của người dùng để khai thác dữ liệu thì không phải chỉ ra tên của đoạn chứa dữ liệu cần lấy.

+ Trong suốt định vị: Trong truy vấn của người dùng để khai thác dữ liệu thì người dùng phải chỉ ra tên của đoạn dữ liệu chứa dữ liệu cần lấy.

+ Trong suốt ánh xạ địa phương: Trong truy vấn của người dùng để khai thác dữ liệu bắt buộc người dùng phải chỉ ra tên của đoạn dữ liệu cần lấy và tên site lưu trữ đoạn dữ liệu đó.

1.2.4.2. Trong suốt giao tác.

- Khái niệm giao tác: Bao gồm nhiều phép toán(Select, Insert, Update, Delete …) được thực hiện trên nhiều bản sao dữ liệu.

- Trong suốt giao tác bao gồm truy vấn phân tán và giao tác phân tán.

+ Truy vấn phân tán: Là truy vấn đến các dữ liệu ở các đoạn dữ liệu khác nhau.

+ Giao tác phân tán: Là bao gồm nhiều lệnh được thực hiện trên nhiều site dữ liệu cùng một lúc.

1.2.4.3. Trong suốt hư hỏng.

Các đoạn dữ liệu được định vị (luu trữ) ở các trạm làm việc khác nhau (có thể mỗi đoạn dữ liệu được định vị tại một trạm hoặc là nhiều đoạn dữ liệu được định vị tại một trạm hoặc là một hay ngiều đoạn dữ liệu đuwocj định vị tên nhiều trạm) tùy thuộc vào nhu cầu phân tán dữ liệu. Nếu dữ liệu trên một trạm bị hỏng thì không làm ảnh hưởng đến các trạm khác. Khi đó truy vấn để lấy dữ liệu sẽ lấy ở những trạm khác nhau.

1.3 HỆ QUẢN TRỊ CƠ SỞ DỮ LIỆU PHÂN TÁN.

1.3.1 Khái niệm HQT-CSDL phân tán.

Hệ quản trị cơ sở dữ liệu phân tán là phần mềm có chức năng quản trị cơ sở dữ liệu phân tán và thực hiện các thao tác trong suốt đến người sử dụng.

1.3.2 Chức năng của HQT-CSDL.

- Nhận, phân tích cú pháp của truy vấn.
- Chuyển đổi truy vấn SQL thành biểu thức đại số quan hệ.
- Tối ưu hóa truy vấn để có cách truy cập hiệu quả nhất. Tức là tối ưu hóa các phép toán đại số quan hệ để có được thời gian xử lý nhanh nhất.
- Phải có các giao diện nhập xuất dữ liệu.
- Định dạng dữ liệu phù hợp với định nghĩa của nó.
- Cung cấp các chức năng cho người quản trị CSDL.

1.3.3 Kiến trúc của HQT-CSDL phân tán.

1.3.3.1. Tính độc lập của dữ liệu.

Cơ sở dữ liệu của một hệ thống luôn được cập nhật trong quá trình vận động của nó. Một tập các thông tin được lưu trữ trong cơ sở dữ liệu tại một thời điểm được gọi là một thể hiện của cơ sở dữ liệu. Thiết kế toàn bộ của cơ sở dữ liệu được gọi là sơ đồ của cơ sở dữ liệu.

Tùy theo mức độ trừu tượng của cơ sở dữ liệu mà sơ đồ của một hệ cơ sở dữ liệu có nhiều mức. Mức thấp nhất được gọi là sơ đồ vật lý, sơ đồ này định nghĩa cấu trúc của dữ liệu. Mức cao hơn là sơ đồ logic xác định cấu trúc logic của dữ liệu.

Tính độc lập của dữ liệu tương ứng gồm 2 mức:

- Độc lập về mặt vật lý: là khả năng khi có sửa đổi sơ đồ vật lý thì các chương trình ứng dụng không cần phải viết lại.
- Độc lập về mặt logic: là khả năng khi có sửa đổi sơ đồ logic thì các chương trình ứng dụng không cần phải viết lại.

Khái niệm độc lập dữ liệu tương tự như khái niệm về các kiểu dữ liệu trừu tượng trong ngôn ngữ lập trình. Chúng ẩn đi các chi tiết thực hiện đối với người sử dụng, mà chỉ cho phép người sử dụng tập trung vào các cấu trúc chung hơn là các chi tiết ứng dụng ở mức thấp.

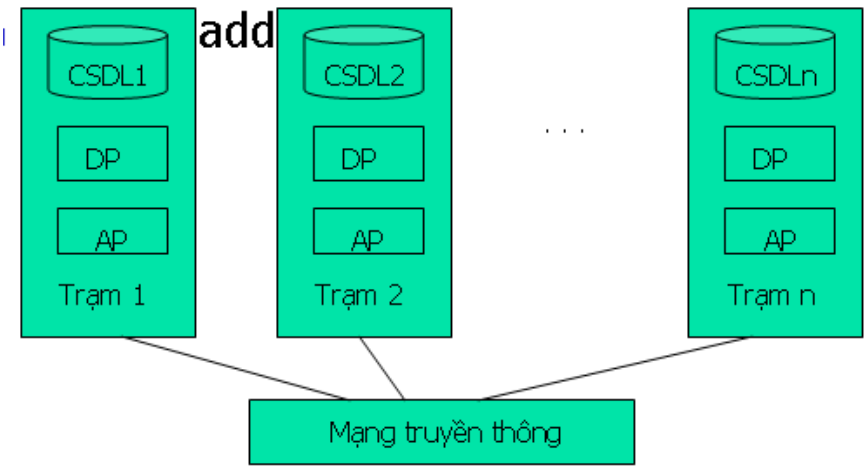
1.3.3.2. Kiến trúc của hệ cơ sở dữ liệu phân tán.

- Có nhiều máy tính được gọi là các trạm (nút – node).
- Các trạm phải được kết nối bởi một kiểu mạng truyền thông để truyền dữ liệu và các lệnh giữa các trạm.
- Phần mềm quản lý hệ cơ sở dữ liệu phân tán:

i, Xử lý dữ liệu (DP – Data Processor): Quản lý dữ liệu cục bộ (địa phương) tại một trạm.

ii. Xử lý ứng dụng (AP – Application Processor): Thực hiện chức năng phân tán, truy cập thông tin phân tán từ thư mục CSDL phân tán và xử lý các yêu cầu truy cập đến nhiều trạm.

iii, Phần mềm truyền thông.

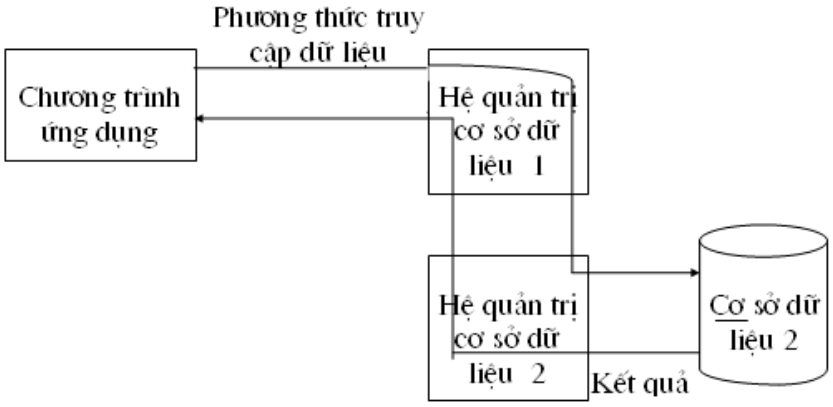


Kiến trúc đơn giản hóa.

1.3.4 Cách thức truy cập cơ sở dữ liệu từ xa

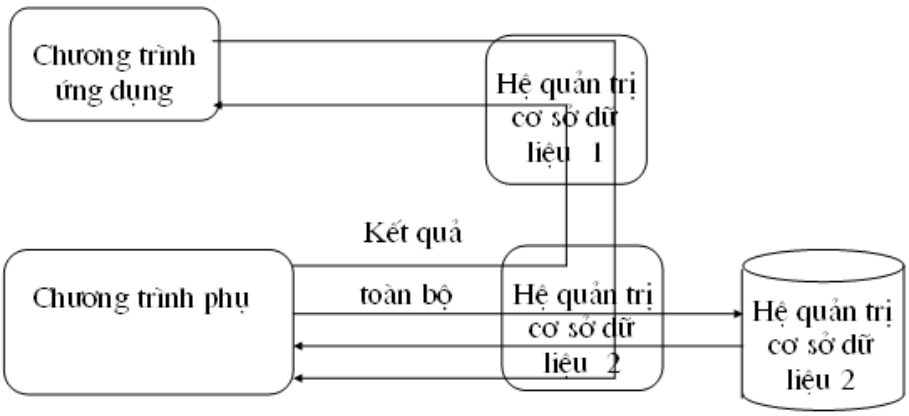
Theo hai cách cơ bản: Truy cập từ xa trực tiếp và gián tiếp.

- Mô hình truy cập từ xa trực tiếp.



Chương trình ứng dụng đưa ra yêu cầu truy cập đến cơ sở dữ liệu từ xa, yêu cầu này được hệ quản trị cơ sở dữ liệu tự động tìm nơi đặt dữ liệu và thực hiện yêu cầu tại điểm đó. Kết quả được trả lại cho trình ứng dụng. Đơn vị chuyển đổi giữa hai hệ quản trị cơ sở dữ liệu là phương thức truy cập cơ sở dữ liệu và kết quả nhận được. Với cách thức truy cập từ xa như vậy cấp độ trong suốt phân tán được xây dựng bằng cách tạo ra tên file toàn bộ để đánh địa chỉ thích hợp cho những điểm lưu trữ dữ liệu ở xa.

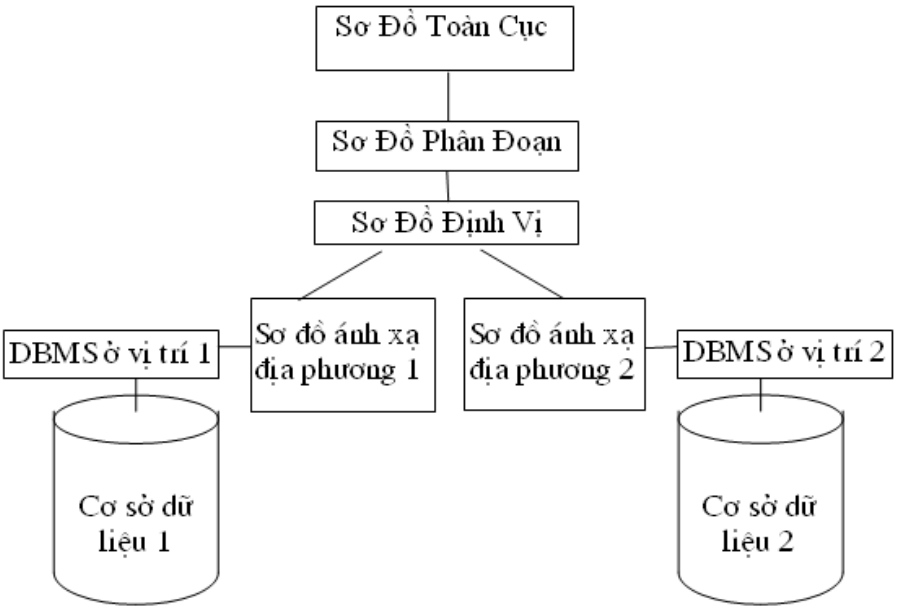
+ Mô hình truy cập từ xa gián tiếp:



Theo mô hình này, chương trình ứng dụng thực hiện yêu cầu qua chương trình phụ ở điểm khác. Chương trình phụ này được người lập trình ứng dụng viết để truy cập từ xa đến cơ sở dữ liệu và trả về kết quả của chương trình ứng dụng yêu cầu.

Hệ quản trị cơ sở dữ liệu phân tán cung cấp cả hai kiểu truy cập.

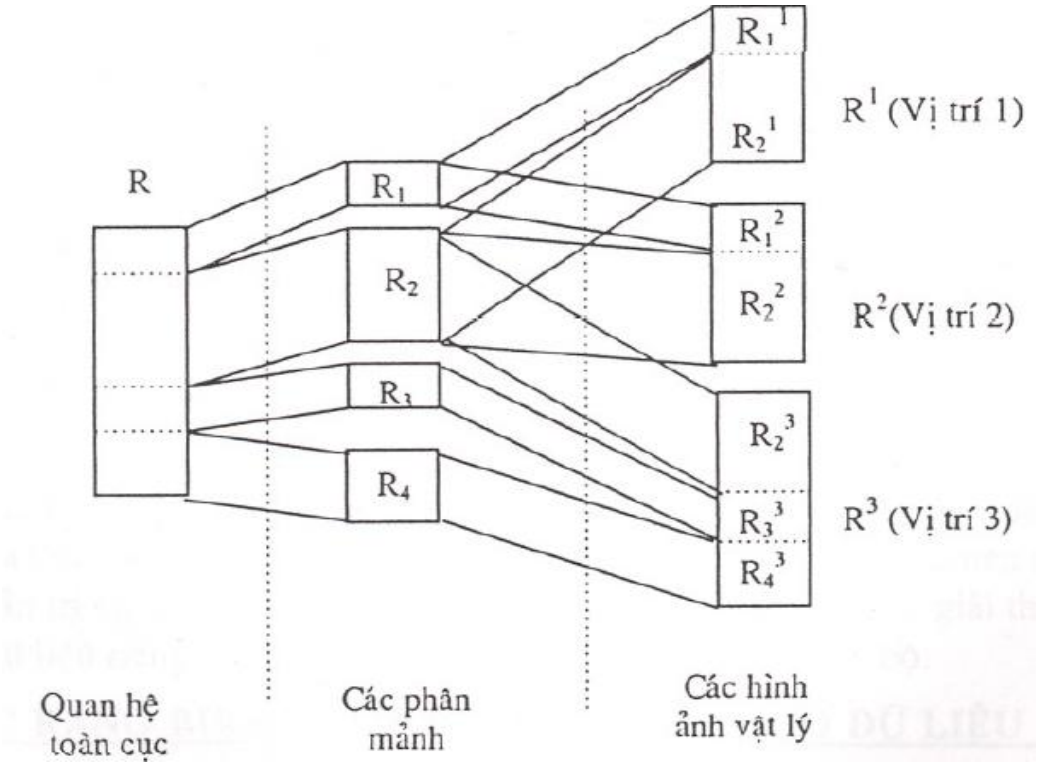
1.3.5 Cấu trúc tham khảo của hệ cơ sở dữ liệu phân tán.



Cấu trúc tham khảo của hệ cơ sở dữ liệu phân tán

Hình trên minh họa một cấu trúc tham khảo của cơ sở dữ liệu phân tán. Cấu trúc này không phải trong mọi trường hợp đều được cài đặt tường minh trong tất cả cơ sở dữ liệu phân tán. Tuy nhiên các mức của nó là khái niệm thích hợp để có thể hiểu được sự tổ chức của mỗi cơ sở dữ liệu.

- Sơ đồ toàn cục: Xác định tất cả dữ liệu được chứa trong cơ sở dữ liệu phân tán.
- Sơ đồ phân đoạn: Mỗi quan hệ toàn cục có thể chia ra thành nhiều phần không chồng lấp lên nhau, gọi là các phân mảnh. Có thể thực hiện việc phân mảnh theo nhiều cách. Việc ánh xạ giữa các quan hệ toàn cục và các phần phân mảnh được gọi là sơ đồ phân mảnh.
- Sơ đồ định vị: Các phân mảnh của các quan hệ toàn cục được lưu trữ tại một hay nhiều vị trí trên mạng. Sơ đồ định vị định nghĩa vị trí nào sẽ chứa các phân đoạn nào. Với kiểu của ánh xạ được định nghĩa trong sơ đồ định vị sẽ xác định là cơ sở dữ liệu phân tán là lưu trữ dạng có nhiều bản sao hay không bản sao. Trường hợp có nhiều bản sao thì ánh xạ là n-1. Trường hợp có 1 bản sao thì ánh xạ là 1-1. Các phân đoạn của cùng một quan hệ toàn cục và sự định vị của nó tại vị trí j trên mạng được gọi là hình ảnh vật lý của quan hệ toàn cục R tại vị trí j. Vì vậy có 1 ánh xạ 1-1 giữa một hình ảnh vật lý và một cặp (quan hệ toàn cục, vị trí) và các hình ảnh vật lý có thể được xác định bởi 1 tên quan hệ toàn cục và một chỉ mục của vị trí. Ký hiệu R_i để chỉ phân đoạn thứ i và R^j để chỉ hình ảnh vật lý của quan hệ toàn cục R tại vị trí j.



Các phân đoạn và hình ảnh vật lý của một quan hệ toàn cục

CHƯƠNG 2. GIỚI THIỆU GIAO TÁC PHÂN TÁN.

2.1. Khái niệm giao tác.

Một giao tác là một đơn vị chương trình được thực hiện nhằm mục đích truy xuất các đơn vị dữ liệu có thể được lưu trữ tại nhiều vị trí khác nhau.

Giao tác có thể thực hiện việc đọc, ghi, tính toán tạo ra dữ liệu mới cho cơ sở dữ liệu, vì vậy các yêu cầu của giao tác là tính nhất quán và tin cậy.

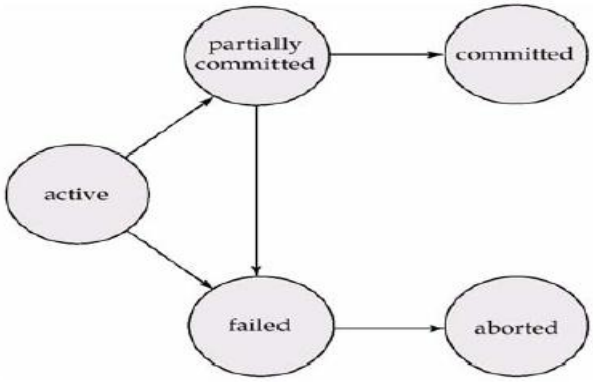
Một giao tác gồm các câu lệnh (thao tác). Để truy xuất đến cơ sở dữ liệu, giao tác có thể thực hiện các thao tác sau:

- Read(X): Đọc đơn vị dữ liệu X từ cơ sở dữ liệu vào trong vùng đệm cục bộ mà giao tác này có thể đọc được.
- Write(X): Giao tác viết đơn vị dữ liệu X từ vùng đệm cục bộ vào trở lại cơ sở dữ liệu.

2.2 Các trạng thái của giao tác.

Dựa vào mức độ hoàn thành các lệnh của giao tác mà chia giao tác thành các trạng thái khác sau:

- Active: Trạng thái đang hoạt động.
- Partially commited: Đã xác nhận từng phần.
- Failed: Sau khi phát hiện ra việc thực hiện một cách bình thường là không thể tiếp tục.
- Aborted: Sau khi giao tác khôi phục lại giống như trạng thái trước khi giao tác bắt đầu.
- Commited: Sau khi giao tác đã hoàn tất.



Biểu đồ trạng thái tương ứng của một giao tác.

Trong trường hợp không có hỏng hóc xảy ra thì các lệnh của giao tác sẽ lần lượt được thực hiện hoàn toàn. Nếu giao tác phải bỏ qua giữa chừng thì mọi thay đổi trên cơ sở dữ liệu mà giao tác đang thực hiện dở phải được trả về trạng

thái ban đầu và khi đó ta phải quay trở lại từng bước để khôi phục dữ liệu về trạng thái cũ.

- Một giao tác sau khi thực hiện hoàn tất được gọi là có trạng thái commited. Giao tác ở trạng thái này sẽ thực hiện cập nhật và biến đổi cơ sở dữ liệu sang một trạng thái nhất quán mới, và trạng thái này vẫn được duy trì trong trường hợp hệ thống bị hỏng hóc. Một giao tác ở trạng thái commited sẽ không thể rơi vào trạng thái Aborted và ngược

Một yêu cầu đặt ra là trong trường hợp có hỏng hóc xảy ra thì không được gây lên mất mát dữ liệu do giao tác đang thực hiện giữa chừng.

Một giao tác trong trạng thái Failed sau khi hệ thống xác định rằng giao tác việc tiếp tục thực hiện là không thể (có thể do lỗi phần cứng hoặc lỗi logic …). Vì vậy sau khi hệ thống khôi phục ta phải chọn 1 trong 2 truờng hợp sau:

- *Restar* lại giao tác này: Có lỗi về phần cứng hoặc phần mềm mà không thể xử lý bởi lỗi bên trong của giao tác. Việc restar xem như thực hiện một giao tác mới.

- *Undo* lại giao tác này: Có lỗi bên trong của giao tác hoặc là dữ liệu không tìm thấy trong cơ sở dữ liệu.

Chúng ta thận trọng trong trường hợp khi ghi dữ liệu ra thiết bị ngoài như máy in, trong trường hợp này không thể *Undo* được. Vì vậy, chỉ ghi ra thiết bị ngoài khi giao tác trong trạng thái commited. Một cách giải quyết khác là có thể viết tạm vào bộ nhớ ngoài, sau khi hoàn tất thì mới in ra giấy. Trong trường hợp này, nếu hệ thống bị hỏng sau khi hoàn tất nhưng chưa kịp ghi ra thiết bị ngoài thì hệ cơ sở dữ liệu sẽ thực hiện việc này khi hệ thống được *Restar* lại.

Trong những ứng dụng có tính chắc chắn, đòi hỏi phải thể hiện dữ liệu cho người sử dụng, đặc biệt là những giao tác thực hiện trong một khoảng thời gian dài, chúng ta không thể xuất dữ liệu giữa chừng được.

2.3 Các thuộc tính của giao tác.

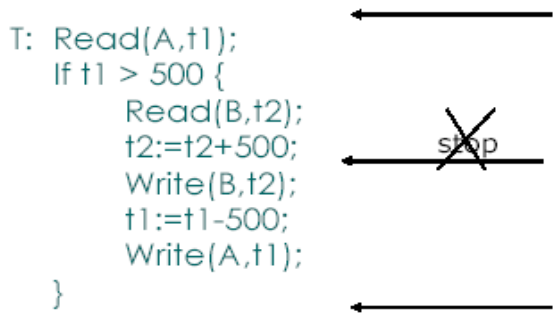
Để đảm bảo tính toàn vẹn dữ liệu đòi hỏi các giao tác phải có một số các thuộc tính:

2.3.1 Tính Nguyên tử (Atomicity).

Hoặc là tất cả các thao tác của một giao tác phải được thực hiện đem lại kết quả đúng đắn, hoặc là không có thao tác nào được thực hiện. Tính nguyên tử đòi hỏi nếu 1 giao tác bị hủy giữa chừng thì các kết quả trước đó của nó phải được hủy bỏ.

Có 2 nguyên nhân làm cho một giao tác phải huy bỏ: Lỗi do chính giao tác này gây nên như: lỗi do dữ liệu nhập vào,…hoặc lỗi của hệ thống như lỗi thiết bị, lỗi do đường truyền, do mất điện…

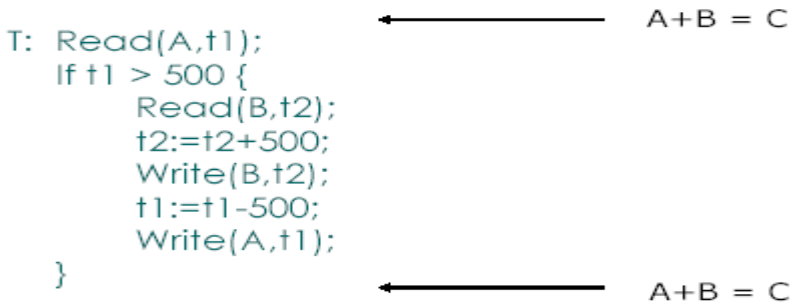
Ví dụ:



2.3.2 Tính nhất quán(Consistency).

Tính nhất quán của dữ liệu trước khi bắt đầu và sau khi kết thúc giao dịch. Tính nhất quán của giao tác được hiểu là nó làm cho dữ liệu được đúng đắn. Để thỏa mãn tính chất này đòi hỏi các giao tác phải chuyển cơ sở dữ liệu từ vị trí nhất quán này đến vị trí nhất quán khác.

Ví dụ:



Với khái niệm dữ liệu tạm là các giá trị dữ liệu được viết bởi một giao tác trong khoảng thời gian nó đang thực hiện. Phân loại 4 cấp độ nhất quán như sau:

- Cấp 0: Giao tác T được gọi là nhất quán cấp 0 nếu T không viết đè lên dữ liệu tạm của giao tác khác.
- Cấp 1: Giao tác T được gọi là nhất quán cấp 1 nếu:

  T là nhất quán cấp 0

  T không thực xác nhận bất kỳ một thao tác ghi nào trước khi kết thúc giao tác (EOT - end of transaction).
- Cấp 2: Giao tác T được gọi là nhất quán cấp 2 nếu:

  T là nhất quán cấp 1.

  T không đọc dữ liệu tạm từ giao tác khác.
- Cấp 3: giao tác T được gọi là nhất quán cấp 3 nếu:

  T là giao tác nhất quán cấp 2.

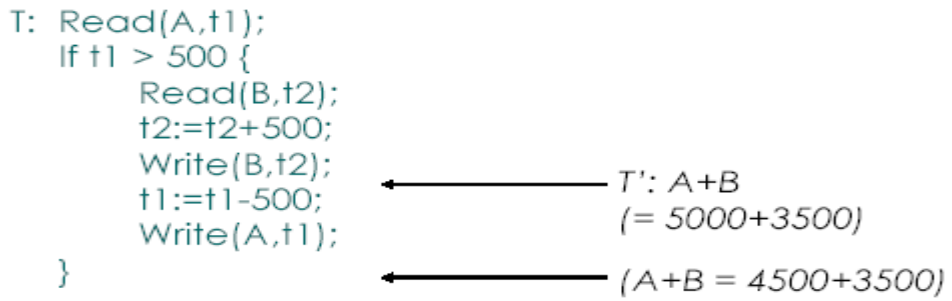
Các giao tác không thực hiện thao tác thay đổi bất kỳ dữ liệu nào đọc bởi T trước khi T xác nhận.

2.3.3 Tính cô lập (Isolation).

Tính cô lập yêu cầu mỗi giao tác phải kiểm tra điều kiện nhất quán tại mọi thời điểm. Hay một giao tác đang thực hiện chưa hoàn tất thì không đưa kết quả của nó cho một giao tác tương tranh khác trước khi nó hoàn tất, nghĩa là nếu có 2 giao tác tương tranh với nhau cùng truy xuất 1 đơn vị dữ liệu, và một trong chúng đã cập nhật thì phải bảo đảm rằng giao tác kia đọc một giá trị đúng.

Ví dụ:

A = 5000, B = 3000



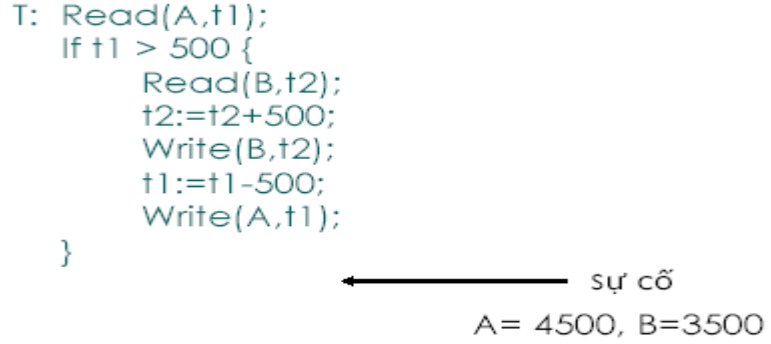
Tính chất này nhằm giải quyết vấn để mất mát khi cập nhật dữ liệu. Ngoài ra còn giải quyết vấn đề hủy bỏ dây truyền. Nếu một giao tác A chưa hoàn tất cho B đọc kết quả của nó thì nếu A hủy bỏ thì kéo theo B hủy bỏ và có thể kéo theo các giao tác khác hủy bỏ.

2.3.4 Tính bền vững (Durability).

Mọi thay đổi mà giao tác thực hiện trên cơ sở dữ liệu phải được ghi nhận bền vững.

Ví dụ:

A = 50000, B = 3000



Sau khi một giao tác thực hiện xong thì những thay đổi của nó trên một cơ sở dữ liệu vẫn được duy trì, thậm chí trong trường hợp hệ thống bị hỏng hóc.

CHƯƠNG 3: TƯƠNG TRANH VÀ CẬP NHẬT DỮ LIỆU

3.1 TỔNG QUAN VỀ TƯƠNG TRANH.

3.1.1 Vì sao phải thực hiện tương tranh.

Việc cho thực hiện các giao tác cập nhật dữ liệu tương tranh với nhau khá phức tạp với yêu cầu bảo đảm tính nhất quán của cơ sở dữ liệu. Bảo đảm sự nhất quán dữ liệu mà không quan tâm tới sự thực hiện tương tranh các giao dịch sẽ cần thêm các công việc phụ. Một phương pháp dễ tiến hành là cho các giao tác thực hiện tuần tự: đảm bảo rằng một giao tác khởi động chỉ sau khi giao tác trước đã hoàn tất. Tuy nhiên tương tranh là vấn đề cần thiết bởi vì các lý do sau:

- Tận dụng tài nguyên: Giao tác này thao tác trên CPU, giao tác khác thực hiện việc đọc hoặc ghi đĩa. Một giao tác gồm nhiều bước. Một vài bước liên quan tới hoạt động I/O, các bước khác liên quan đến hoạt động của CPU. CPU và các đĩa trong một hệ thống có thể hoạt động song song với xử lý tại CPU. Sự song song của hệ thống và CPU và I/O có thể được khai thác để chạy nhiều giao tác song song. Trong khi một giao tác tiến hành một hoạt động đọc/viết trên một đĩa, một giao tác khác có thể chạy trong CPU, một giao tác thứ 3 có thể thực hiện đọc/viết trên một đĩa khác … Như vậy sẽ tăng lượng đầu vào hệ thống có nghĩa là tăng số lượng giao tác có thể thực hiện trong một lượng thời gian đã cho, cũng có nghĩa là hiệu suất sử dụng CPU và đĩa tăng lên.

- Giảm thời gian đáp ứng trung bình: Nếu cho thực hiện tuần tự thì 1 giao tác tuy ngắn nhưng có thể phải đợi 1 giao tác khác cho đến khi hoàn tất rất lâu do đó không thể đoán trước được thời gian đợi là bao lâu. Có thể trộn lẫn các giao dịch đang chạy trong hệ thống, có giao tác dài, giao tác ngắn nếu thực hiện tuần tự, một quá trình ngắn có thể phải chờ một quá trình dài đến trước hoàn tất, mà điều đó dẫn đến một sự trì hoãn không lường trước được trong việc chạy một giao tác. Nếu các giao tác đang hoạt động trên các phần khác nhau của cơ sở dữ liệu, sẽ tốt hơn nếu ta cho chúng chạy đồng thời, chia sẻ các chu kỳ CPU và truy xuất đĩa giữa chúng. Thực hiện tương tranh làm giảm sự trì hoãn không lường trước trong việc chạy các giao tác đồng thời giảm thời gian đáp ứng trung bình: thời gian để một giao dịch được hoàn tất sau khi đã được đệ trình.

Khi có nhiều giao tác thực hiện tương tranh với nhau thì yêu cầu đặt ra là các thuộc tính của giao tác là phải thỏa mãn. Khi đó tính nhất quán có thể bị phá hủy cho dù mỗi giao tác là đúng. Một giải pháp để giải quyết vấn đề này là sử dụng định thời.

Ví dụ:

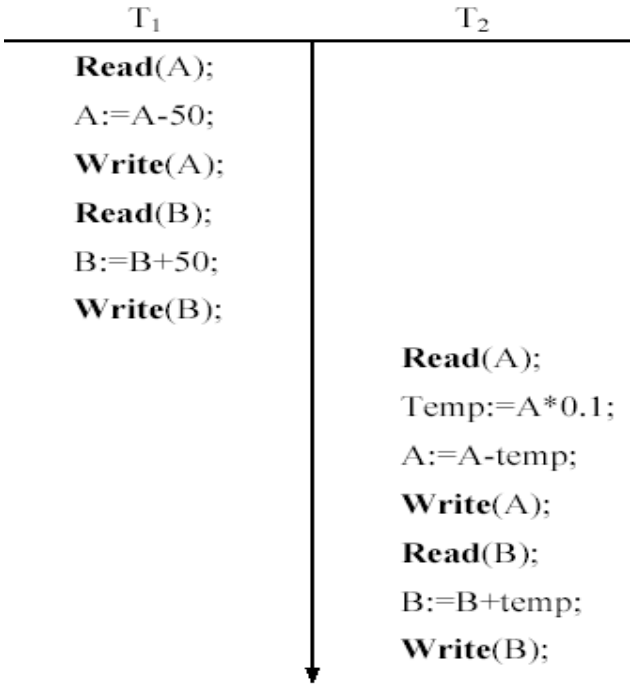
Xét hệ thống ngân hàng đơn giản, nó có một số tài khoản và có một tập hợp các giao tác, chúng truy xuất, cập nhật các tài khoản này. Giả sử giao tác T_1 chuyển 50$ từ tài khoản A sang tài khoản B:

```
T1:   Read(A);
      A:=A-50;
      Write(A);
      Read(B);
      B:=B+50;
      Write(B);
```

Giao tác T_2 chuyển 10% số dư tài khoản A sang B:

```
T2:   Read(A);
      Temp:=A*0.1;
      A:=A-Temp;
      Write(A);
      Read(B);
      B:=B+Temp;
      Write(B);
```

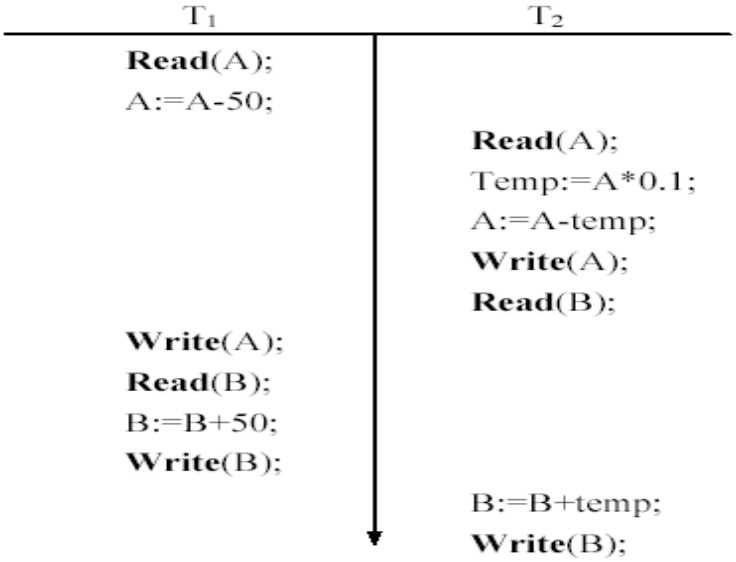
Giả sử giá trị hiện thời của A = 500$ và B = 1000$. Giả sử rằng 2 giao tác này thực hiện theo thứ thự T_1 rồi đến T_2.



Giá trị sau cùng của A và B sau khi thực hiện dãy các chỉ thị theo trình tự là A = 405$ và B = 1095$. Như vậy tổng giá trị của hai tài khoản này bảo tồn sau khi thực hiện giao tác.

Dãy thực hiện vùa mô tả trên đươc gọi là các lịch trình. Chúng biểu diễn trình tự thời gian các chỉ thị được thực hiện trong hệ thống. Có thể có một vài

dãy thực hiện, vì nhiều chỉ thị của các giao dịch có thể đan xen nhau. Không phải tất cả các thực hiện cạnh tranh đều cho ra trạng thái đúng. Ta xét Ví dụ sau:



Ta thấy rằng sau giao tác này cuối cùng A = 450$, B = 1050$. Trạng thái này là một trạng thái không nhất quán.

Đây là một lịch trình cạnh tranh.

3.1.2 Tính khả tuần tự.

Định nghĩa: cho n giao tác T_1 , T_2 ,.., T_n

- Ta gọi một lịch S của một tập các giao tác T_1 , T_2 , …, T_n là một thứ tự mà trong đó các lệnh của giao tác này được thực hiện lần lượt hoàn toàn. Ký hiệu: S(T_1 , T_2, …, T_n)

- Một lịch S gọi là tuần tự nếu tất cả các bước của mỗi giao tác đều được thực hiện liên tiếp nhau. Như vậy trong lịch tuần tự mỗi giao tác thực hiện toàn bộ các lệnh của nó và không có tương tranh.

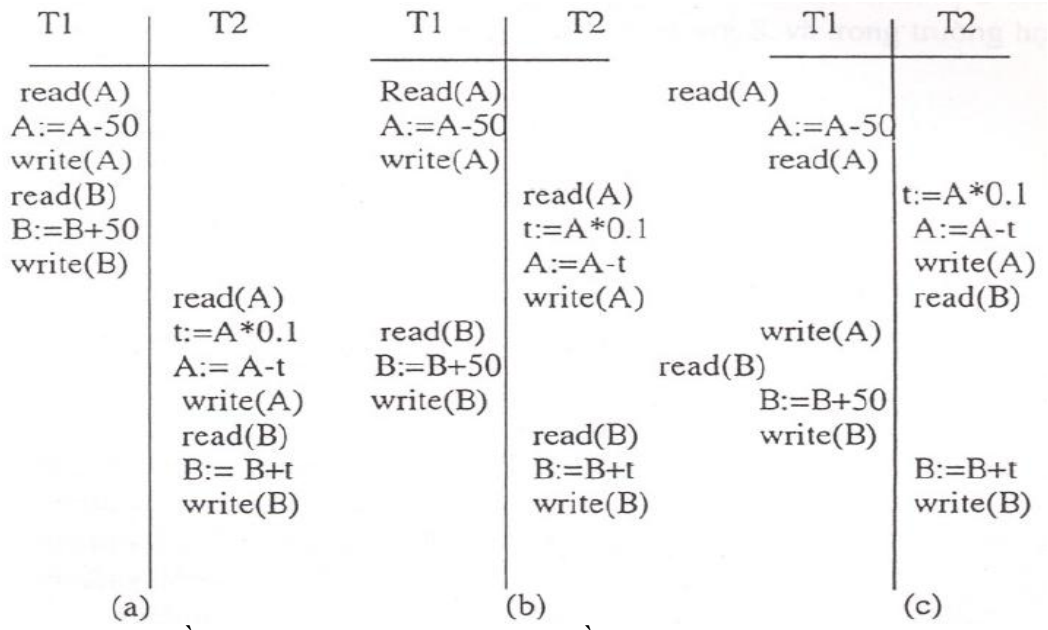
- Một lịch S được gọi là khả tuần tự nếu nó tương đương với 1 lịch tuần tự nào đó (Ta gọi 2 lịch tương đương nếu kết quả cuối cùng trong cơ sở dữ liệu sau khi thực hiện 2 lịch này là như nhau).

Ví dụ: Giả sử công ty có 2 phòng A và B. Phòng A có 1000 nhân viên và phòng B có 2000 nhân viên. Giao tác T_1 chuyển 50 nhân viên từ A sang B và giao tác T_2 chuyển 10% từ A sang B. Ta có các giao tác như sau:

T_1: Read(A)
A:=A-50
Write(A)
Read(B)
B:=B+50
Write(B)
T_2: Read(A)

t:=A*0.1
A:=A-t
Write(A)
Read(B)
B:=B+t
Write(B)

Xét 3 lịch sau:



S_1: lịch tuần tự S_2: lịch khả tuần tự S_3: không khả tuàn tự

S_1 là tuần tự: $S(T_1;T_2)$ và sau khi cho thực hiện tương tranh S_2 thì kết quả tương đương lịch S_1. Sau khi thực hiện lịch S_3 thì giá trị của A = 950 và B= 2000. Rơi vào trạng thái không nhất quán.

Khi thực hiện tương tranh các giao tác phải đảm bảo cơ sở dữ liệu giữ nguyên trạng thái nhất quán. Trước hết ta tìm hiểu lịch nào đảm bảo tính nhất quán và lịch nào không. Sau đây ta xét 2 loại lịch tương đương về tính khả tuần tự đó là: khả tuần tự đụng độ và khả tuần tự quan sát.

3.1.2.1 Khả tuần tự xung đột.

Xét lịch S có 2 lệnh I_i và I_j lần lượt của 2 giao tác T_i và T_j .

Nếu 2 lệnh này tham khảo đến các mục dữ liệu khác nhau thì có thể đổi chỗ T_i và T_j mà không ảnh hưởng đến bất kỳ lệnh nào trong lịch trình. Nếu 2 lệnh này mà cùng tác động đến 1 mục Q thì thứ tự là rất quan trọng. Ta có 4 trường hợp sau:

- I_i = read(Q), I_j = read(Q): Thứ tự là không quan trọng.
- I_i = read(Q), I_j = write(Q): Nếu I_i trước I_j thì T_i không đọc giá trị viết bởi T_j và nếu I_j đến trước I_i thì I_i đọc giá trị bởi T_j . Như vậy thứ tự là quan trọng.

- I_i = write(Q), I_j = read(Q): Tương tự như trên thứ tự là quan trọng.

- I_i = write(Q), I_j = write(Q): Nếu chỉ có hai lệnh này thì thứ tự là không quan trọng. Tuy nhiên nó tác động đến lệnh read(Q) kế tiếp, nó sẽ đọc giá trị của lệnh được viết sau cùng.

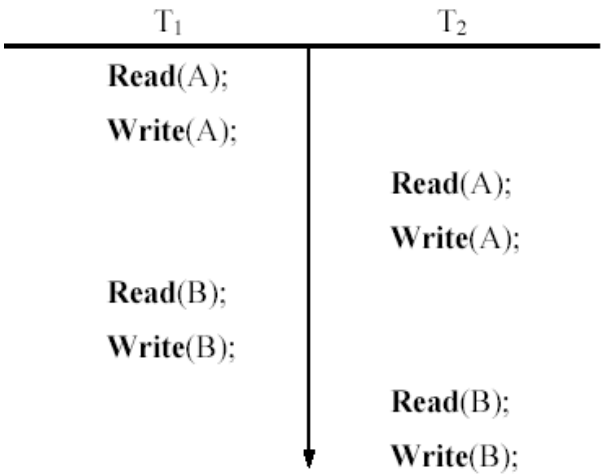
Ta nói là hai lệnh là xung đột nếu nó chúng là các lệnh của các giao tác khác nhau tác động lên cùng một đơn vị dữ liệu và trong chúng có ít nhất một thao tác write.

Nhận xét:

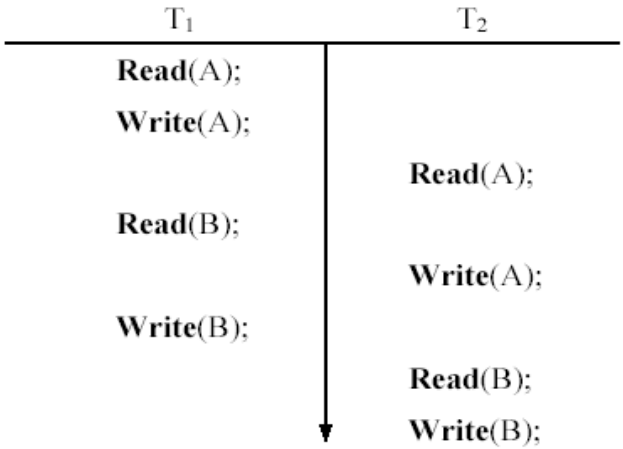
- Nếu 2 lệnh không xung đột nhau ta có thể hoán vị 2 lệnh này với nhau để tạo nên 1 lịch S' tương đương xung đột với S.

Ví dụ:

Xét lịch S sau:



Vì write(A) của T_2 không xung đột với read(B) của T_1, ta có thể đổi chỗ các lệnh này để được một lịch trình tương đương.



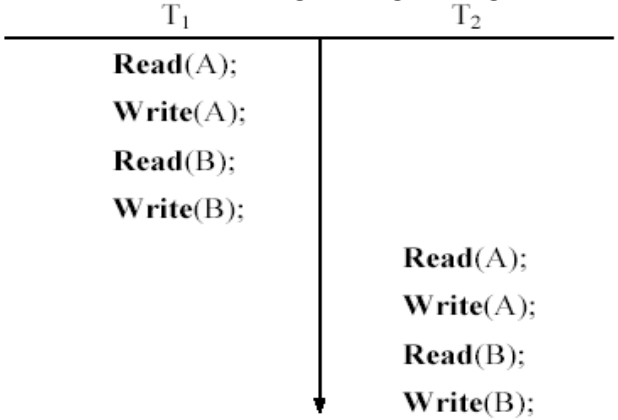
Ta tiếp tục đổi chỗ các chỉ thị không xung đột sau:

+ Lệnh read(B) của T_1 với lệnh read(A) của T_2 .

+ Lệnh write(B) của T_1 với lệnh write(A) của T_2 .

+ Lệnh write(B) của T_1 với lệnh read(A) của T_2 .

Ta sẽ có lịch trình mới tương đương đuơng với lịch trình ban đầu.



Khái niệm tuần tự xung đột: Ta nói một lịch S là khả tuần tự xung đột nếu nó tương đương xung đột với một lịch trình tuần tự.

3.1.2.2 Khả tuần tự quan sát.

Cho 2 lịch S và S' trên cùng một tập các giao tác T_1, T_2 , … . T_n . Ta gọi S và S' tương đương quan sát nếu thỏa 3 điều kiện sau:

1. Với mỗi đơn vị dữ liệu Q, nếu T_i nhận giá trị khởi trị của Q trong lịch S thì T_i cũng phải nhận giá trị khởi trị của Q trong S'.
2. Với mỗi đơn vị dữ liệu Q, nếu trong lịch S giao tác T_i đọc giá trị của Q mà giá trị này được xử lý bởi T_j, thì trong lịch S' giao tác T_i cũng phải đọc giá trị của Q mà giá trị này được xử lý bởi T_j .
3. Với mỗi đơn vị dữ liệu Q, nếu trong lịch S giao tác T_i thực hiện thao tác ghi sau cùng , thì trong lịch S' giao tác T_i cũng phải thực hiện thao tác ghi sau cùng.

Điều kiện 1 và 2 đảm bảo mỗi giao tác đọc cùng các giá trị trong cả hai lịch trình và do vậy thực hiện cùng tính toán. Điều kiện 3 đảm bảo cả hai lịch trình cho ra kết quả là trạng thái cuối cùng của hệ thống như nhau.

Các lịch khả tuần tự xung đột thì khả tuần tự quan sát, tuy nhiên không có điều ngược lại.

3.1.3 Các lịch có khả năng khôi phục dữ liệu.

Phần trên ta xét các lịch bảo đảm cho sự nhất quán của cơ sở dữ liệu và giả sử không xảy ra trường hợp các giao tác hỏng hóc. Xét trường hợp xảy ra một giao tác nào đó bị hỏng trong quá trình thực hiện tương tranh. Nếu một giao tác T_i nào đó bị hỏng thì cần thiết hủy bỏ những gì giao tác này thực hiện để đảm bảo tính nguyên tử. Mặt khác trong hệ cho phép thực hiện tương tranh thì phải

bảo đảm rằng mọi giao tác phụ thuộc T_i (chẳng hạn: T_j đọc dữ liệu đã được T_i ghi) cũng phải được hủy bỏ. Để thỏa mãn được điều kiện này ta cần thiết giới hạn các loại của lịch. Trong phần điều khiển tương tranh chúng ta chỉ xét các lịch chấp nhận được.

Sau đây ta xét 2 lịch thỏa mãn điều kiện trên.

3.1.3.1 Lịch khả phục hồi.

Khái niệm: Một lịch trình khả phục hồi là lịch trình trong đó, đối với mỗi cặp giao dịch T_i , T_j nếu T_j đọc mục dữ liệu được viết bởi T_i thì hoạt động bàn giao của T_j phải xảy ra sau hoạt động bàn giao của T_i .

Ta xét Ví dụ sau:

T_1	T_2
read(A)	
write(A)	
	read(A)
read(B)	

T_2 là một giao tác chỉ thực hiện một chỉ thị read(A). Giả sử cho phép T_2 bàn giao (commit) ngay sau khi thực hiện lệnh read(A). Như vậy T_1 bàn giao trước T_1. Giả sử T_1 thất bại trước khi bàn giao. Vì T_2 đọc giá trị thất bại của T_1 được viết bởi T_1 nên ta phải bỏ dở T_2 để đảm bảo tính nguyên tử. xong T_1 đã được bàn giao và không thể hủy bỏ được. Ta không thể khôi phục đúng sau thất bại của T_1.

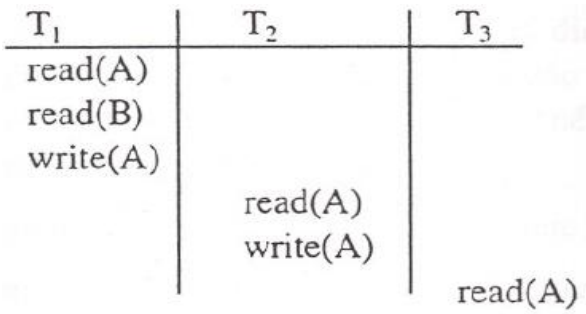
Đây là một lịch trình không thể phục hồi được và không được phép. Các hệ cơ sở dữ liệu phân tán đều có yêu cầu là các lịch đều có khả năng khôi phục được.

3.1.3.2 Lịch không gây hủy bỏ dây chuyền(hủy bỏ Domino).

Định nghĩa: một lịch không gây hủy bỏ dây chuyền là một lịch trong đó mỗi cặp giao tác T_i , T_j , nếu T_j đọc một mục dữ liệu được viết trước đó bởi T_i thì hoạt động bàn giao của T_i phải xảy ra trước hoạt động đọc của T_j .

Nếu một lịch có thể khôi phục được từ việc thất bại của một giao tác T_i nào đó thì có thể xảy ra trường hợp phải cuộn lại nhiều giao tác. Đặc biệt là giao tác đã đọc giá trị viêt bởi T_i .

Ta xét trường hợp sau:



Ta thấy, T_1 ghi giá trị mà T_2 sẽ đọc, sau đó T_2 sẽ ghi giá trị mà T_3 đọc. Giả sử T_1 bị thất bại khi đó T_1 phải cuộn lại, vì T_2 phụ thuộc vào T_1 nên T_2 cũng phải cuộn lại, và vì T_3 lại phụ thuộc vào T_2 nên T_3 cũng phải cuộn lại.

Đây là hiện tượng khi một giao tác nào hủy bỏ thì nó kéo theo hàng loạt các giao tác khác hủy bỏ theo.

Một lịch có khả năng tạo ra việc hủy bỏ dây chuyền là điều không mong muốn. Một lịch là không tạo nên hủy bỏ dây chuyền thì nó cũng là lịch có thể khôi phục được.

3.2 CÁC PHƯƠNG PHÁP ĐIỀU KHIỂN TƯƠNG TRANH TRONG CƠ SỞ DỮ LIỆU PHÂN TÁN.

Một trong các tính chất cơ bản của giao tác là tính độc lập. Khi một vài giao tác thực hiện một cách tương tranh tính độc lập có thể không được bảo tồn. Đối với hệ thống cần phải điều khiển sự trao đổi giữa các giao dịch tương tranh, sự điều khiển này được thực hiện thông qua sơ đồ điều khiển tương tranh. Các sơ đồ điều khiển tương tranh dựa trên tính khả tuần tự.

Hiện nay có một số thuật toán được cung cấp liên quan đến việc điều khiển tương tranh. Các thuật toán điều khiển tương tranh theo 2 lớp:

- Các thuật toán trên cơ sở khóa (Lock) dữ liệu là độc quyền hay chia sẻ.
- Các thuật toán dựa theo thứ tự thực hiện của các giao tác theo các giao thức.

3.2.1 Các phương pháp điều khiển tương tranh phân tán trên cơ sở khóa.

3.2.1.1 Tổng quan về khóa.

Một phương pháp để đảm bảo tính tuần tự là yêu cầu việc truy xuất đến hạng mục dữ liệu được tiến hành theo kiểu loại trừ tương hỗ. Có nghĩa là trong khi một giao dịch đang truy xuất một hạng mục dữ liệu, không một giao tác nào

khác có thể sửa đổi hạng mục này. Phương pháp chung nhất được dùng để thực thi yêu cầu này là cho phép một giao tác truy xuất một mục dữ liệu chỉ nếu nó đang giữ khóa trên mục dữ liệu này.

Tư tưởng chính của các thuật toán này là các thao tác trên một đơn vị dữ liệu nếu có xung đột thì chỉ cho phép một giao tác thực hiện tại một thời điểm. Điều này được thực hiện dựa trên việc khóa đơn vị dữ liệu.

Để truy xuất một mục dữ liệu, giao tác T_i đầu tiên phải khóa hạng mục này. Nếu hạng mục này đã bị khóa bởi một giao tác khác ở phương thức không tương thích, bộ điều khiển tương tranh sẽ không cấp khóa cho đến tận khi tất cả các khóa không tương thích bị giữ bởi các giao tác khác được tháo. Như vậy T_i phải chờ đến tận khi tất cả các khóa không tương thích bị giữ bởi các giao tác khác được giải phóng.

Giao tác T_i có thể tháo khóa mục dữ liệu mà nó đã khóa trước đây. Một giao tác cần thiết phải giữ một khóa trên một mục dữ liệu chừng nào mà nó còn truy xuất mục này. Hơn nữa, đối với một giao tác việc tháo khóa ngay sau khi truy xuất cuối cùng đến mục dữ liệu không luôn luôn là điều mong muốn vì như vậy tính khả tuần tự có thể không được đảm bảo.

Ta xét Ví dụ sau:

Xét hoạt động tại một công ty sau: có 2 phòng A và B.

- Giao tác T_1 chuyển 50 nhân viên từ phòng B sang phòng A. Ta minh họa như sau:

```
T1 :   Lock-X(B);
       Read(B);
       B:=B-50;
       Write(B);
       Unlock(B);
       Lock-X(A);
       Read(A);
       A:=A+50;
       Write(A);
       Unlock(A);
```

- Giao tác T_2 hiển thị tổng số nhân viên tại 2 phòng. Được minh họa như sau:

T_2 : **Lock-S(A);**
Read(A);
Unlock(A);
Lock-S(B);
Read(B);
Unlock(B);
Display(A+B);

Giả sử A có 100 nhân viên và B có 200 nhân viên. Nếu 2 giao tác này thực hiện một cách tuần tự hoặc thứ tự T_1, T_2 hoặc T_2 , T_1 khi đó T_2 sẽ hiện thị giá trị 300 nhân viên.

Nếu 2 giao tác này thực hiện tương tranh như sau:

T_1	T_2	Bộ quản trị điều khiển cạnh tranh
Lock-X(B)		
		Grant-X(B,T_1)
Read(B)		
B:=B-50		
Write(B)		
Unlock(B)		
	Lock-S(A)	
		Grant-S(A,T_2)
	Read(A)	
	Unlock(A)	
	Lock-S(B)	
		Grant-S(B,T_2)
	Read(B)	
	Unlock(B)	
	Display(A+B)	
Lock-X(A)		
		Grant-X(A,T_1)
Read(A)		
A:=A+50		
Write(A)		
Unlock(A)		

Trong trường hợp này giao tác T_2 sẽ hiển thị giá trị 250 nhân viên một kết quả không đúng. Lý do của sai lầm này là giao tác T_1 đã tháo khóa mục B quá sớm và T_2 tham chiếu một trạng thái không nhất quán.

Các hành động được thực hiện bởi các giao tác cũng như các thời điểm khi các khóa được cấp bởi bộ điều khiển tương tranh. Giao tác đưa ra một yêu cầu khóa không thể thực hiện hành động kế tiếp của mình đến tận khi khóa được cấp bởi bộ điều khiển tương tranh. Do đó khóa phải được cấp trong khoảng thời gian giữa hoạt động yêu cầu khóa và hành động sau của giao tác.

Trong phương pháp này, sự đồng bộ của các giao tác đạt được bằng cách thực hiện việc chiếm giữ vật lý hoặc là chiếm giữ logic trên các phần nhỏ của cơ sở dữ liệu. Dựa vào việc quản lý việc khóa dữ liệu mà các thuật toán bao gồm:

- Quản lý khóa tập trung: Một trong các vị trí trên mạng được thiết kế như là một vị trí trung tâm, tại vị trí này lưu trữ bảng khóa của toàn bộ cơ sở dữ liệu và được giao nhiệm vụ cấp phát việc chiếm giữ dữ liệu cho các giao tác.

- Quản lý khóa của bản sao chính: Khi cơ sở dữ liệu được thiết kế theo kiểu bản sao, trong đó một bản sao được thiết kế như là một bản sao chính. Một giao tác nào đó muốn khóa một đơn vị dữ liệu nào của cơ sở dữ liệu thì trước hết phải được phép khóa tại bản sao chính này.

Ví dụ: X có 3 bản sao vị trí 1, vị trí 2 và vị trí 3. Giả sử vịt rí 2 được chọn làm vị trí chính cho X. Như vậy bất kỳ giao tác nào muốn truy xuất đến một bản sao nào đó của X thì phải khóa X tại vị trí 2 trước.

- Quản lý khóa phân tán: Việc quản lý khóa được chia sẻ cho các vị trí trên mạng. việc thực hiện các giao tác phụ thuộc vào các lịch của các vị trí điều phối và các lịch của các vị trí thành viên.

Khi 1 giao tác thực hiện việc truy xuất một đơn vị dữ liệu thì trước hết phải xin khóa dữ liệu. Các phương thức khóa mục dữ liệu:

▪ Shared (S) hay ReadLock(RL): nếu một giao tác T_i nhận được một khóa ở phương thức shared trên mục Q, khi đó T_i có thể đọc nhưng không được viết Q.

▪ Exclusive (X) hay WriteLock(WL): nếu một giao tác T_i nhận được một khóa ở phương thức WL, khi đó T_i có thể cả đọc và viết.

Khái niệm tương thích giữa các phương thức: 2 phương thức khóa là tương thích với nhau nếu chúng có thể thực hiện đồng thời trên 1 đơn vị dữ liệu.

Mỗi giao tác đòi hỏi một khóa ở một phương thức thích hợp trên một mục dữ liệu, phương thức này phụ thuộc vào kiểu hoạt động mà nó sẽ thực hiện trên mục dữ liệu đó. Quan hệ tương thích giữa hai phương thức khóa được cho bởi ma trận comp sau:

	RL	WL
RL	True	False
WL	False	False

Comp(A,B) = True => các phương thức A và B tương thích.

Comp(A,B) = False => các phương thức A và B không tương thích.

Một giao tác yêu cầu khóa trên mục Q bằng cách thự hiện lệnh:

- lock-S(Q) hoặc Rlock(Q): yêu cầu khóa theo phương thức RL.
- lock-X(Q) hoặc Wlock(Q): yêu cầu khóa theo phương thức WL.

- unlock(Q): yêu cầu tháo khóa.

3.2.1.2 Một số tình huống không mong đợi.

3.2.1.2.1 Tình trạng DeadLock

Ta xét lịch sau:

T_3	T_4
Lock-X(B)	
Read(B)	
B:=B-50	
Write(B)	
	Lock-S(A)
	Read(A)
	Lock-S(B)
Lock-X(A)	

Do T_3 giữ khóa phương thức WL trên B, nên yêu cầu một khóa phương thức RL của T_4 trên B phải chờ đến khi T_3 tháo khóa. Cũng như vậy, T_3 yêu cầu một khóa WL trên A trong khi T_4 đang giữ một khóa RL trên nó và như vậy phải chờ. Ta gặp phải tình huống trong đó T_3 chời đợi T_4 đồng thời T_4 chờ đợi T_3 dẫn đến sự chờ đợi vòng tròn và như vậy không giao tác nào có thể tiến triển. Tình huống này gọi là *DeadLock* (khóa chết). Khi tình huống khóa chết xảy ra hệ thống buộc phải cuộn lại một trong các giao tác. Mỗi khi một giao tác bị cuộn lại, các mục dữ liệu bị khóa bởi giao tác phải được tháo khóa và nó trở nên sẵn sàng cho giao tác khác, như vậy các giao tác này có thể tiếp tục được sự thực hiện của nó.

Nếu ta không sử dụng khóa hoặc tháo khóa mục dữ liệu ngay khi có thể sau đọc hoặc viết mục dữ liệu ta có thể rơi vào trạng thái không nhất quán. Mặt khác nếu ta không tháo khóa một mục dữ liệu trước khi yêu cầu một khóa trên mục khác thì deadlock có thể xảy ra. *Deallock* là khó tránh khi sử dung khóa.

3.2.1.2.2 Tình trạng LiveLock.

Xét trường hợp sau: Giả sử T_2 đang khóa phương thức RL, T_1 có yêu cầu khóa phương thức WL, do đó T_1 phải đợi đến khi T_2 giải phóng, trong thời gian này T_3 có yêu cầu khóa phương thức RL vì tương thích với T_2 nên được cho phép, T_1 vẫn đợi, và lại có T_4 có yêu cầu khóa phương thức RL, … T_1 vẫn phải

đợi và không được thực hiện cho đến khi T_2 , T_3 , T_4 , … tháo khóa. Tình trạng này được gọi là *LiveLock*. Để tránh trường hợp này, giải quyết như sau:

Khi một giao tác T_i có yêu cầu khóa đơn vị dữ liệu X, thì T_i sẽ được phép nếu:

- Không có một giao tác nào khác khóa X với phương thức không tương thích.

- Không có một giao tác nào khác đang đợi để khóa X mà trước T_i .

Ta sẽ yêu cầu mỗi giao tác trong hệ thống tuân theo một tập các quy tắc, được gọi là giao thức khóa (locking protocol), chỉ định một giao tác có thể khóa và tháo khóa mỗi một trong các mục dữ liệu. Giao thức khóa hạn chế số các lịch trình có thể. Tập các lịch trình như vậy là một tập con thực sự của tập tất cả các lịch trình khả tuần tự có thể.

Xét { T_0 , T_1 , …, T_n } một tập các giao tác tham gia vào lịch trình S. ta nói T_i đi trước t_j trong S, và được viết là: T_i -> T_j , nếu tồn tại một mục dữ liệu Q sao cho T_i giữ khóa phương thức A trên Q, T_j giữ khóa phương thức B trên Q muộn hơn và comp(A,B) = false. Nếu T_i -> T_j thì T_i sẽ xuất hiện trước T_j trong bất kỳ lịch trình tuần tự nào. Ta nói một lịch S là hợp lệ dưới một giao thức khóa nếu S là một lịch trình tuân thủ các quy tác giữ khóa của phương thức khóa đó.

3.2.1.3 Phương pháp khóa 2 pha.

Phương pháp khóa 2 pha là một phương pháp đảm bảo tính khả tuần tự. Phương pháp này yêu cầu mỗi một giao tác phát ra yêu cầu khóa và tháo khóa thành 2 pha riêng biệt:

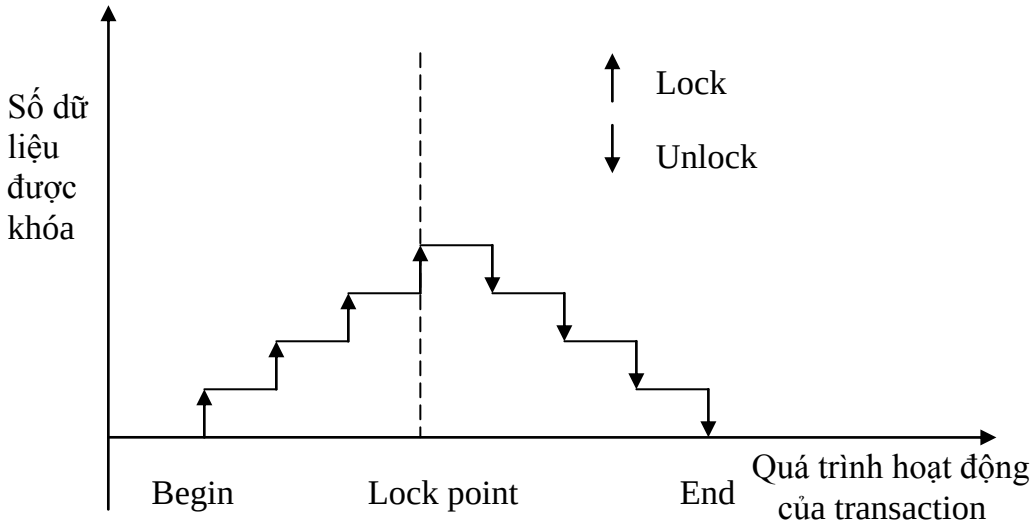
1. Pha tăng trưởng hay pha xin khóa (Growing phase): Một giao tác có thể nhận được các khóa, nhưng nó không thể tháo bất kỳ khóa nào (cho phép lock mà không cho phép unlock).
2. Pha co giảm hay pha tháo khóa (Shrinking phase): Một giao tác có thể tháo các khóa nhưng không thể nhận được một khóa mới nào (cho phép unlock mà không cho lock mới).

Khởi đầu một giao tác ở pha xin khóa. Giao tác xin được rất nhiều khóa cần thiết. Mỗi khi giao tác tháo một khóa, nó đi vào kỳ tháo khóa và nó không thể phát ra bất kỳ một yêu cầu xin khóa nào nữa.

Trong phương pháp khóa 2 pha khi một giao tác T_i nào đó tháo khóa đơn vị dữ liệu X thì nó cho phép các lệnh của giao tác này liền sau đó khóa ngay một đơn vị dữ liệu khác. Điều này có khả năng nâng cao khả năng tương tranh, nó cho phép giao tác này xen vào giao tác khác, tuy nhiên nó làm mất đi tính riêng biệt và tính nguyên tử.

Phương pháp khóa 2 pha quy định không có một giao tác nào khóa sau khi nó đã tháo khóa. Như vậy giao tác không được tháo khóa cho đến khi chắc chắn không còn yêu cầu khóa nào nữa.

Thời điểm trong lịch cuối cùng của pha tăng trưởng được gọi là điểm khóa (*lock point*) của giao tác. Các giao tác có thể sắp thứ tự theo các thời điểm khóa của chúng, đây cũng là thứ tự tuần tự của các giao tác.

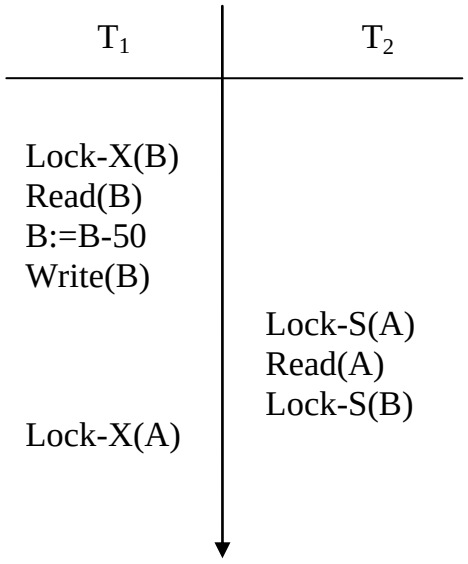


Sơ đồ khóa 2 pha.

Nhược điểm của phương pháp khóa 2 pha:

- Phương pháp khóa 2 pha đòi hỏi phải biết được tất cả các yêu cầu khóa của mỗi giao tác và thời điểm bắt đầu tháo khóa.
- Phương pháp khóa 2 pha không bảo đảm tránh được *deadlock* và việc cuộn lại hàng loạt.

Ta xét Ví dụ sau vẫn thỏa phương pháp khóa 2 pha nhưng vẫn rơi vào tình trạng *deadlock*.



Giao tác T_1 đang trong pha tăng trưởng yêu cầu khóa mục dữ liệu B theo phương thức WL, xử lý bớt 50 dữ liệu trên B và yêu cầu khóa mục dữ liệu A theo phương thức WL. Nhưng vì giao tác T_2 cũng đang trong pha tăng trưởng và đang giữ khóa mục A theo phương thức RL 2 phương thức này không tương thích nhau nên bộ điều phối sẽ không cấp phát. Cũng như vậy giao tác T_2 xin khóa mục B theo phương thức không tương thích nên T_2 cũng bị treo dẫn đến T_1 và T_2 đều bị treo dẫn đến *deadlock*.

⇨ Đây là nhược điểm lớn của phương khóa 2 pha. Để khắc phục tình trạng này có phương pháp khóa 2 pha ngiêm ngặt.

Phương pháp khóa 2 pha nâng cấp chuyển đổi khóa.

Để nâng cao khả năng tương tranh sự cải tiến của phương pháp khóa 2 pha cho phép chuyển đổi nâng cấp giữ các kiểu khóa: nâng cấp một khóa RL sang WL và hạ cấp một khóa WL thành RL. Sự nâng cấp chỉ được phép diễn ra trong pha tăng trưởng, hạ cấp chỉ được diễn ra trong pha co giảm.

- Nâng cấp (upgrade): chuyển từ kiểu khóa RL thành kiểu khóa WL.
- Hạ cấp (downgrade): chuyển từ kiểu khóa WL thành kiểu khóa RL.

Việc nâng cấp trong pha tăng trưởng và hạ cấp trong pha co giảm thì tính khả tuần tự vẫn không thay đổi.

Mệnh đề:

Nếu các giao tác của một lịch S đều thỏa phương pháp khóa 2 pha và có thực hiện nâng cấp trong pha tăng trưởng, hạ cấp trong pha co giảm thì tính khả tuần tự của một lịch vẫn không đổi.

Xét 2 giao tác sau:

T_1 : read(A_1)
read(A_2)
…
read(A_1)

write(A_1)

T_2: read(A_1)

read(A_2)

dísplay(A_1 + A_2)

Nếu ta sử dụng phương pháp khóa 2 pha, khi đó T_1 phải khóa A_1 theo kiểu WL. Bởi vậy, sự thực hiện tương tranh của 2 giao tác rút cuộc trở thành thực hiện tuần tự. Ta thấy rằng T_1 chỉ cần khóa WL trên A_1 chỉ ở cuối sự thực hiện của nó, khi nó write(A_1). Như vậy T_1 có thể khởi động khóa ở phương thức RL và đổi sang phương thức WL sau này. Như vậy ta có thể nhận được sự tương tranh cao hơn vì T_1 và T_2 có thể truy xuất đến A_1 và A_2 đồng thời như hình dưới đây:

T_1	T_2
Lock-S(A_1)	
	Lock-S(A_2)
Lock-S(A_2)	
	Lock-S(A_2)
Lock-S(A_3)	
...	
	Unlock(A_1)
	Unlock(A_2)
UpGrade(A_1)	

Sơ đồ đơn giản nhưng được sử dụng rộng rãi để sinh tự động các chỉ thị khóa và tháo khóa thích hợp cho một giao tác: mỗi khi giao tác T xuất ra một lệnh read(Q), hệ thống sẽ xuất ra một lệnh lock-S(Q) ngay trước lệnh read(Q). Mỗi khi giao tác T xuất ra một hoạt động write(Q), hệ thống sẽ kiểm tra xem T đã giữ một khóa RL nào trên Q hay chưa:

- Nếu đã, nó xuất ra một chỉ thị upgrade(Q) ngay trước lệnh write(Q).
- Nếu chưa, nó xuất ra lệnh lock-X(Q) ngay trước lệnh write(Q).

Tất cả các khóa giao dịch nhận được sẽ được tháo khóa sau khi giao tác bàn giao hay bỏ dở.

Ghi chú: Nếu một giao tác phải hảy bỏ và hồi phục lại trạng thái ban đầu sau khi đã tháo khóa thì có thể kéo theo các giao tác khác có truy xuất vào các dữ liệu dã mở khóa này hủy bỏ theo. Vì vậy trong phương pháp khóa 2 pha có thể xảy ra tình trạng cuộn lại hàng loạt.

3.2.1.4 Phương pháp khóa 2 pha nghiêm ngặt.

Nếu một giao tác phải hủy bỏ sau khi đã tháo khóa thì có thể kéo theo các giao tác khác có thể truy xuất vào các dữ liệu đã tháo khóa này dẫn đến hủy bỏ theo. Vì vậy trong phương pháp khóa 2 pha có thể xảy ra tình trạng cuộn lại hàng loạt.

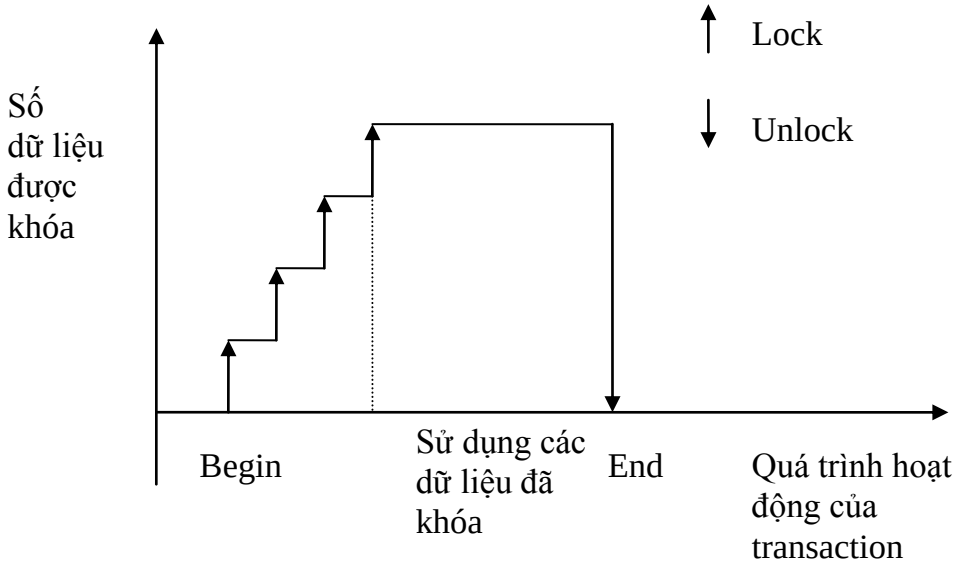
Ta xét Ví dụ sau:

T_1	T_2	T_3
Lock-X(A) Read(A) Lock-S(B) Read(B) Write(A) Unlock(A)		
	Lock-X(A) Read(A) Write(A) Unlock(A)	
		Lock-X(A) Lock-S(A)

Ta thấy nếu T_1 bị lỗi sau khi read(A) thì dẫn đến cuộn lại cả T_2 và T_3. Có thể tránh cuộn lại hàng loạt bằng cách sửa đổi phương pháp khóa 2 pha thành phương pháp khóa 2 pha nghiêm ngặt (S2LP: strict 2-3 phase locking protocal).

Phương pháp khóa 2 pha nghiêm ngặt đòi hỏi tất cả các khóa phương thức WL phải được giữ đến tận khi giao dịch bàn giao. Yêu cầu này đảm bảo rằng bất kỳ dữ liệu nào được viết bởi một giao dịch chưa bàn giao bị khóa phương thức WL đến tận khi được bàn giao, điều này ngăn chặn bất kỳ giao tác khác đọc dữ liệu này vì có thể dẫn đến trạng thái không nhất quán.

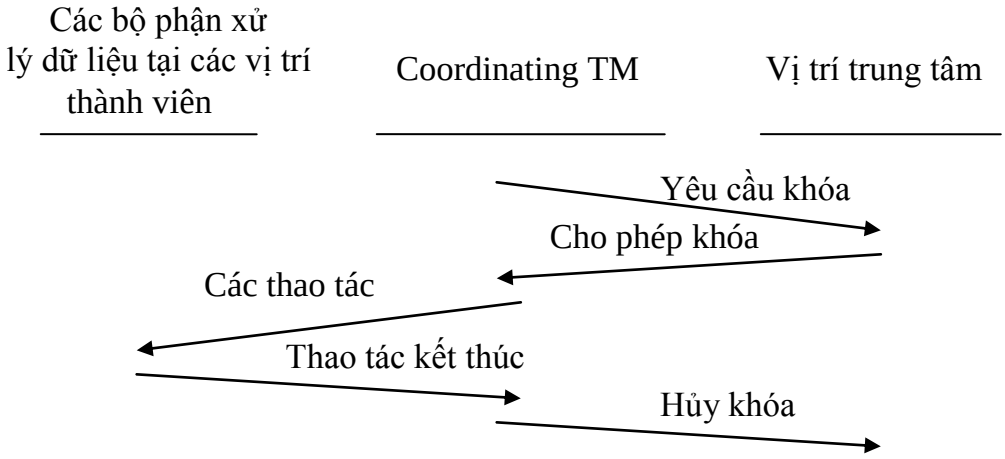
Hầu hết các hệ cơ sở dữ liệu đều là áp dụng phương pháp khóa 2 pha nghiêm ngặt.



Sơ đồ khóa 2 pha nghiêm ngặt.

3.2.1.5. Phương pháp khóa 2 pha trung tâm (Centralized 2PL).

Giao trách nhiệm quản lý khóa cho cho một vị trí nào đó. Điều này có nghĩa là chỉ một vị trí nào đó bộ phận quản lý khóa (LM: lock manager), các bộ phận quản lý giao tác (TM: Transaction manager) tại các vị trí khác phải liên lạc với nó, vị trí này được gọi là vị trí trung tâm. Trong phương pháp này có sự liên lạc giữa bộ phận quản lý giao tác tại vị trí giao tác được khởi tạo (TM điều phối: coordinating TM) với bộ phận quản lý khóa tại vị trí trung tâm và bộ phận xử lý dữ liệu tại vị trí thành viên khác. Bộ phận quản lý khóa trung tâm không gửi các thao tác đến bộ xử lý dữ liệu cho từng vị trí mà thông qua bộ phận quản lý giao tác điều phối.



Sơ đồ liên lạc trong phương pháp C2PL

Nhược điểm:

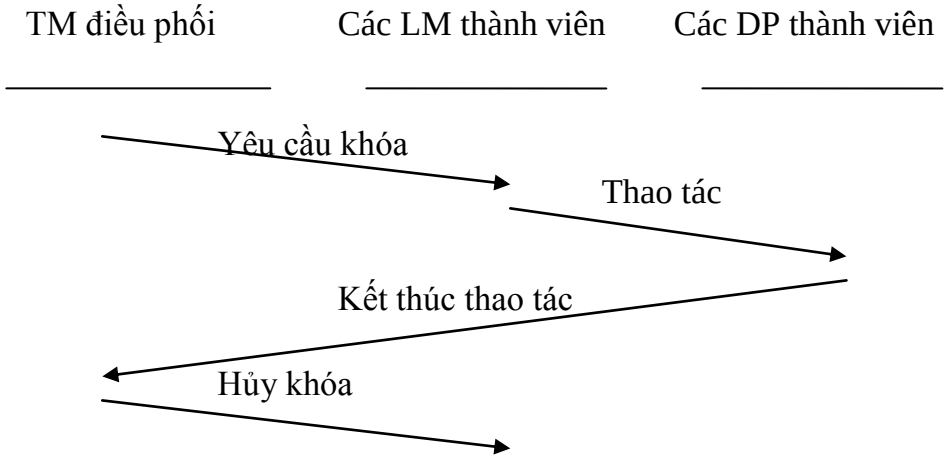
- Có thể xảy ra tình trạng thắt cổ chai khi có nhiều giao tác cùng truy xuất đến vị trí trung tâm.
- Độ tin cậy của hệ thống không cao nếu vị trí trung tâm bị hỏng.

3.2.1.6 Phương pháp khóa 2 pha phân tán (D2LP-TM).

Tại mỗi vị trí đều có bộ phận quản lý khóa LM.

Phương pháp khóa tương tự phương pháp khóa 2 pha trung tâm nhưng khác 3 điểm sau:

- Trong phuơng pháp khóa 2 pha trung tâm các thông báo được gửi đến vị trí trung tâm (là bộ phận quản lý khóa), còn trong phương pháp khóa 2 pha phân tán các thông báo được gửi đến bộ phận quản lý khóa của tất cả các vị trí thành viên.
- Trong phương pháp khóa 2 pha trung tâm các thao tác được chuyển đến các bộ phận xử lý giao tác điều phối, còn trong phương pháp khóa 2 pha phân tán các thao tác được chuyển đến các LM thành viên. Điều này có nghĩa là LM điều phối không đợi thông báo: yêu cầu khóa được cho phép.
- Thành viên gửi thông báo: kết thúc thao tác cho TM điều phối thay vì mỗi DP phải gửi cho LM của nó để được cho phép hủy bỏ khóa và thông báo cho TM điều phối.



Sơ đồ truyền thông trong phương pháp khóa 2 pha phân tán

3.2.2 Điều khiển tương tranh dựa trên nhãn thời gian.

Phương pháp này chọn lựa một thứ tự của các giao tác theo cách giao tác nào đến trước hơn gọi là điều khiển tương tranh theo nhãn thời gian.

3.2.2.1 Thuật toán thứ tự nhãn thời gian.

Nhãn thời gian.

Mỗi giao tác T_i trong hệ thống, ta sẽ gán cho nó một nhãn thời gian (Timestamp) duy nhất là $TS(T_i)$. Thòi nhãn này được gán bởi hệ cơ sở dữ liệu trước khi T_i được thực hiện. Nếu có một giao tác mới T_j trong hệ thống thì $TS(T_i) < TS(T_j)$. $TS(T_i)$ được gán bằng cách:

- Dùng giá trị của đồng hồ hệ thống: giá trị của nhãn thời gian được gán bằng giá trị của đồng hồ khi giao tác gia nhập vào hệ thống.
- Dùng phép đếm logic nó được tăng lên khi 1 nhãn thời gian mới được gán. Vì vậy nhãn thời gian của một giao tác bằng giá trị đếm khi giao tác gia nhập vào hệ thống.

Một đơn vị dữ liệu có các giá trị nhãn thời gian sau:

- WTS(Write-Timestamp): Nhãn thời gian lớn nhất của các giao tác sau khi thực hiện thành công write(Q).
- RTS(Read-Timestamp): Nhãn thời gian lớn nhất của các giao tác sau khi thực hiện thành công read(Q).

Các giá trị này được cập nhật mỗi khi có 1 giao tác mới xin write(Q) hoặc read(Q).

Luật nhãn thời gian: Cho 2 lệnh xung đột O_i và O_j lần lượt của 2 giao tác T_i và T_j , ta nói O_i là được thực hiện trước O_j nếu và chỉ nếu $TS(T_i)<TS(T_j)$.

Thuật toán thứ tự nhãn thời gian: thuật toán bảo đảm cho các thao tác read và write được thực hiện theo thứ tự nhãn thời gian như sau:

1. Nếu giao tác T_i cần read(Q):
   - Nếu $TS(T_i)<WTS(Q)$: T_i đọc giá trị chưa ghi xong, từ chối phép đọc này và T_i phải cuộn lại.
   - Nếu $TS(T_i)>=WTS(Q)$: cho phép đọc và $RTS(Q):=max(RTS(Q),TS(T_i))$.
2. Nếu giao tác T_i cần write(Q):
   - Nếu $TS(T_i)<RTS(Q)$: T_i phải cuộn lại.
   - Nếu $TS(T_i)<WTS(Q)$: T_i phải cuộn lại.
   - Còn lại cho phép write(Q) và $WTS(Q):=TS(T_i)$.

(Cuộn lại là gán cho nó một nhãn thời gian mới và restart lại).

Ví dụ: hai giao tác T_1 và T_2 được xác định như sau:

T_1: Read(B);
Read(A);
Display(A+B)

T_2: Read(B)
B:=B-50
Write(B)
Read(A)
A:=A+50
Write(A)
Display(A+B)

⇨ Giả sử $TS(T_1)<TS(T_2)$ lịch S là một lịch trình hợp lệ giao thức nhãn thời gian.

T_1	T_2
Read(B)	
	Read(B)
	B:=B-50
	Write(B)
Read(A)	
	Read(A)
Display(A+B)	
	A:=A+50
	Write(A)
	Display(A+B)

Nhận xét:

- Khi lệnh bị từ chối thì sẽ được *restart* và gán một nhãn mới -> các giao tác sẽ được thực hiện sau đó -> tránh được *deadlock* tuy nhiên có thể xảy ra *restart* nhiều lần.
- Bảo đảm nó khả tuần tự đụng độ vì các lệnh dược xử lý theo thứ tự nhãn thời gian.

Quy tắc ghi Thomas

Cho phép tính tương tranh cao hơn. Ta xét Ví dụ sau:

T_1	T_2
Read(Q)	
	Write(Q)
Write(Q)	

Nếu áp dụng giao thức thứ tự nhãn thời gian, ta có $TS(T_1) < TS(T_2)$. Read(Q) của T_1 và write(Q) của T_2 thành công, Nhưng $TS(T_1) < TS(T_2) = WTS(Q)$ nên write(Q) của T_1 bị vứt bỏ giao dịch T_1 bị cuộn lại. Sự cuộn lại này là không cần thiết

Từ Ví dụ trên ta thấy rằng thao tác ghi có thể bỏ qua trong tình chắc chắn. Một sửa đổi của giao thức tự nhãn thời gian: quy tắc ghi Thomas:

- Quy tắc đối với read là không thay đổi.
- Write được sửa lại 1 chút: nếu $TS(T_i) < WTS(Q)$: T_i đang thử viết 1 giá trị lỗi thời của Q. Do vậy, hoạt động write có thể bị bỏ lơ (không được thực hiện, nhưng T_i không bị cuộn lại).

Luật ghi của Thomas nó sẽ xóa đi các thao tác Write không cần thiết, vì vậy nó có thể tạo ra các lịch khả tuần tự

3.2.2.2 Phương pháp dựa trên tính hợp lệ.

Trong trường hợp phần lớn các giao tác là read thì tỷ lệ đụng độ giữa các giao tác giảm. Trong trường hợp này cần phải giảm đi việc đợi của các giao tác. Ta giả sử rằng mỗi giao tác thực hiện trong 2 pha hoặ 3 pha tùy theo giao tác này chỉ đọc hoặc ghi:

- Kỳ đọc: Các giá trị của các mục dữ liệu khác nhau được đọc vào các biến cụa bộ của T_i . Tất cả các hoạt động write được thực hiện trên các biến cục bộ tạm, không cập nhật vào cơ sở dữ liệu.
- Kỳ hợp lệ: Giao dịch T_i thực hiện một phép kiểm thử sự hợp lệ để xác định xem nó có thể cập nhật vào cơ sở dữ liệu các biến cục bộ tạm chứa các kết quả của các hoạt động write mà không vi phạm tính khả tuần tự xung đột hay không.
- Kỳ ghi: Nếu T_i thành công trong kỳ hợp lệ, được cập nhật vào cơ sở dữ liệu, nếu không bị cuộn lại.

Mỗi giao tác trải qua 3 kỳ trên, tuy nhiên ba kỳ của các giao tác đang thực hiện tương tranh có thể đan xen nhau. Để thực hiện kiểm thử sự hợp lệ ta cần biết khi nào các kỳ khác nhau của giao tác T_i xảy ra. Do vậy ta kết hợp 3 nhãn thời gian:

- start(T_i): thời gian T_i bắt đầu thực hiện.
- validation(T_i): thời gian T_i kết thúc kỳ đọc và khởi động kỳ hợp lệ.
- finish(T_i): thời gian T_i kết thúc kỳ viết.

Xác định thứ tự khả tuần tự bằng thuật toán nhãn thời gian với giá trị nhãn thời gian $TS(T_i)$ = validation(T_i) vì nhãn thời gian của giao tác T_i là thời điểm được cung cấp đáp ứng nhanh giảm tỷ lệ đụng độ giữa các giao tác.

Để kiểm tra tính hợp lệ, tất cả các giao tác thỏa $TS(T_i) < TS(T_j)$ phải thỏa 1 trong 2 điều kiện:

- $finish(T_i) < start(T_j)$: T_i hoàn tất trước khi T_j bắt đầu, do đó thứ tự khả tuần tự được thỏa mãn.
- Tập các đơn vị dữ liệu được viết bởi T_i thì không giao nhau với tập các đơn vị dữ liệu được đọc bởi T_j và T_i hoàn tất kỳ ghi trước khi T_j bắt đầu kỳ hợp lệ ($start(T_j) < finish(T_i) < validation(T_j)$). Đảm bảo cho T_i và T_j không đè lên nhau. Khi các lệnh write của T_i không ảnh hưởng lệnh read của T_j và ngược lại thì thứ tự tuần tự vẫn được duy trì.

Ví dụ:

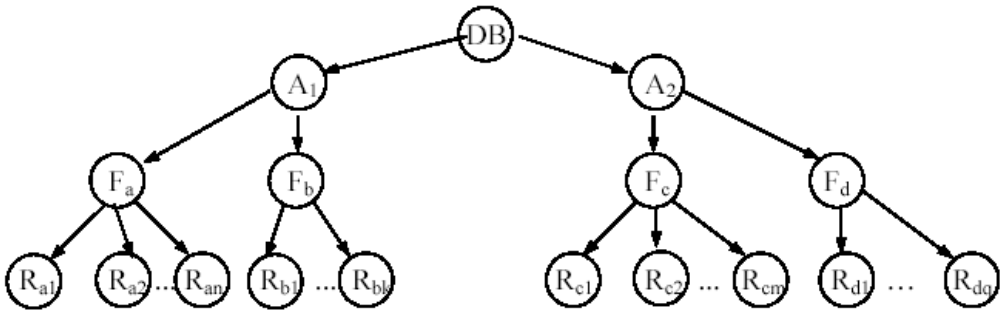
T_1	T_2
Read(B)	
	Read(B)
	B:=B-50
	Read(A)
	A:=A+50
Read(A)	
Xác nhận tính hợp lệ	
Display(A+B)	
	Xác nhận tính hợp lệ
	Write(B)
	Write(A)

Phương pháp này tránh việc cuộn lại hàng loạt do các lệnh write hiện tại xảy ra chỉ sau khi giao tác phát ra write đã bàn giao.

3.2.3 Phương pháp đồ thị.

Chia cơ sở dữ liệu thành các mức khác nhau gọi là các cấp. Sự phân cấp này được biểu diễn bởi đồ thị như là cây dữ liệu.

- Mức cao nhất: toàn bộ cơ sở dữ liệu.
- Mức dưới kế: các node gồm các vùng chứa dữ liệu A_1, A_2
- Mỗi vùng chứa các node con là các tệp tin F_a, F_b , F_c, F_d
- Mỗi tệp tin chứa các record R_{a1}, R_{a1}, … , R_{dq}



Quy tắc khóa:

- Mỗi node có thể khóa riêng một mình. Khi 1 giao tác khóa 1 node bằng 1 kiểu nào đó thì nó cũng khóa tất cả các node con theo kiểu đó nhưng là khóa ẩn.

Ví dụ: T_j muốn khóa R_{b6} của F_b. Nếu T_i chiếm F_6 dạng tường minh thì T_i cũng chiếm R_{b6} dạng ẩn nên T_j phải duyệt từ gốc đến R_{b6} nếu không tương thích thì T_j phải đợi.

- Nếu muốn khóa toàn bộ cơ sở dữ liệu khi đó sẽ khóa gốc của cây. Tuy nhiên phải đảm bảo rằng giao tác đó chưa khóa 1 phần nào của cây. Vấn đề là làm thế nào để biết được có phần nào của cây bị khóa hay chưa. Ta có thể duyệt hoặc dùng kiểu khóa có ý định (IL) (intention lock mode).

IW: khóa có ý định kiểu ghi.

RIW: gốc của cây con bị khóa kiểu read tường minh, 1 node cấp thấp hơn bị khóa kiểu write.

Ma trận tương thích:

	IR	IW	R	RIW	W
IR	T	T	T	T	F
IW	T	T	F	F	F
R	T	F	F	F	F
RIW	T	F	F	F	F
W	F	F	F	F	F

Thuật toán: Nếu lịch S được tạo ra gồm các giao tác thỏa mãn điều kiện sau thì S sẽ khả tuần tự:

- Thỏa ma trận tương thích.
- Gốc của cây được khóa đầu tiên.
- Một node Q có thể bị khóa bởi T_i theo kiểu R hoặc IR chỉ nếu node cha của Q bị khóa bởi T_i theo kiểu IW hoặc IR.

- Một node Q bị khóa bởi T_i theo W,RIW hoặc IW chỉ nếu node cha của Q bị khóa kiểu IW hoặc RIW.

- T_i có thể khóa 1 node nếu trước đó nó chưa unlock một node nào.

- T_i có thể unlock(Q) chỉ nếu không có 1 node con nào của Q là đang bị khóa bởi T_i .

⇨ Lock theo kiểu Top-Down, Unlock theo kiểu Bottom-Up.

Ví dụ:

- Giao tác T_1 cần đọc R_{a2} của F_a , khi đó T_1 khóa A_1 và theo IR và khóa R_{a2} theo kiểu R.

- T_2 cần ghi record R_{a9} của F_a , khi đó sẽ khóa A_1 và F_a theo IW và khóa R_{a9} theo kiểu W

- T_3 cần đọc tất cả các record của F_a , khi đó T_3sẽ khóa A_1 theo IR và khóa F_a theo kiểu R.

- T_4 cần đọc toàn bộ cơ sở dữ liệu, nó sẽ khóa cơ sở dữ liệu theo kiểu R

- Khi đó T_1 , T_3 , T_4 có thể truy xuất tương tranh trên cơ sở dữ liệu này.
- T_2 có thể tương tranh với T_1 nhưng không thể tương tranh với T_3 hoặc T_4

Nhận xét: phương pháp này là sự mở rộng của tính tương tranh của phương pháp khóa 2 pha và giảm đi chi phí khóa dữ liệu. Sử dụng với các ứng dụng có chứa

- Các giao tác có thời gian thực hiện nhanh và truy xuất ít dơn vị dữ liệu.

- Các giao tác có thời gian thực hiện kéo dài tạo ra các report từ toàn bộ cơ sở dữ liệu hoặc tập các tập tin.

3.2.4 Xử lý *deadlock*.

Định nghĩa: hệ thống được gọi là trong tình trạng *deadlock* nếu tồn tại một tập hợp các giao tác mà mỗi giao tác trong tập này đều đợi một giao tác khác dẫn đến không có giao tác nào có thể tiến hành tiếp tục.

Có 2 phương pháp giải quyết tình trạng *deadlock*.

3.2.3.1 Ngăn ngừa deadlock.

Phương pháp này bảo đảm rằng hệ thống không bao giờ xảy ra tình trạng này. Phương pháp này thường được dùng khi xác suất xảy ra trong hệ thống là khá cao.

Có 2 cách tiếp cận để ngăn ngừa *deadlock*:

1. Bảo đảm không để xảy ra chu trình đợi bằng cách cho thực hiện thứ tự của các yêu cầu chiếm giữ hoặc đòi hỏi tất cả các yêu cầu khóa phải được thực hiện đồng thời.

Phương pháp đơn giản nhất của cách này là yêu cầu mỗi giao tác khóa toàn bộ đơn vị dữ liệu trước khi nó thực hiện giao tác. Hơn nữa đòi hỏi mỗi giao tác hoặc là khóa tất cả các đơn vị dữ liệu cần hoặc là không chiếm giữ cái nào.

Một phương pháp khác là cho một thứ tự của các đơn vị dữ liệu tương tự như phần dữ liệu có nhiều cấp, và yêu cầu các giao tác phải khóa theo thứ tự này.

2. Cách tiếp cận này gọi là sử dụng tính chất gọi là chiếm trước (quyền ưu tiên). Để điều khiển quyền ưu tiên này, chúng ta có thể sử dụng nhãn thời gian cho các giao tác. Hệ thống sẽ sử dụng nhãn thòi gian để cho giao tác tiếp tục hoặc cuộn lại.

Hai thuật toán dựa trên cơ sở nhãn thời gian:

- Thuật toán Wait-Die: khi có giao tác T_i yêu cầu khóa dữ liệu Q đang được khóa bởi T_j thì nếu T_i có nhãn thời gian nhỏ hơn T_j thì T_i được phép đợi, ngược lại T_i phải cuộn lại.

- Thuật toán Wound-Wait: khi có giao tác T_i yêu cầu khóa Q đang được giữ bởi T_j thì nếu T_i có nhãn thời gian lớn hơn T_j thì T_i được phép đợi, ngược lại T_j phải cuộn lại.

Trong Wait-Die một giao tác cũ hơn phải đợi giao tác mới hơn tháo dữ liệu, Wound-Wait thì ngược lại. Trong Wait-die nếu T_i phải cuộn lại nấu lần restart tiếp theo nếu dữ liệu đó vẫn bị khóa bởi T_j thì nó vẫn phải cuộn lại. Trong Wound-Wait thì không,

3.2.3.2 Dò tìm và Khôi phục deadlock.

Để dò tìm và khôi phục được deadlock hệ thống phải:

- Duy trì các thông tin về vị trí hiện hành của các đơn vị dữ liệu của các giao tác cũng như các dữ liệu chưa giải quyết xong.
- Cung cấp các thuật toán để khi sử dụng các thông tin này thì xác định hệ thống có rơi vào tình trạng *deadlock* không.
- Khôi phục dữ liệu từ *deadlock* khi xác định được rằng đang xảy ra tình trạng *deadlock*.

Dò Tìm deadlock

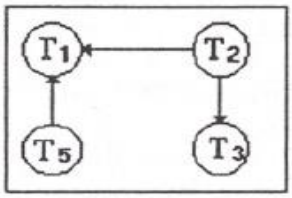
Để dò tìm *deadlock* ta dùng đồ thị có hướng để mô tả được gọi là đồ thị đợi (WFG) gồm 1 cặp G = <V,E> trong đó

V: Tập các đỉnh (các giao tác).

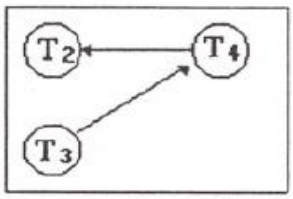
E: Tập các cạnh (mỗi cạnh là một thứ tự T_i -> T_j nói rằng T_i đang đợi khóa đơn vị dữ liệu mà T_j đang khóa).

Deadlock xảy ra khi đồ thị có chu trình. Để khôi phục *deadlock* mỗi một vị trí sẽ lưu trữ một đồ thị đợi cục bộ (LWFG).

Ví dụ:

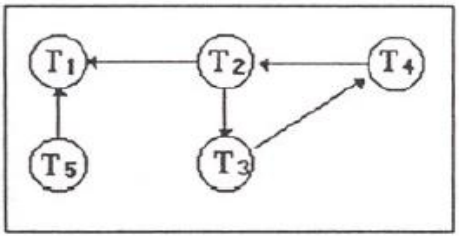


Vị trí 1



Vị trí 2

Khi giao tác T_i tại vị trí S_1 cần tài nguyên của vị trí S_2 thì nó sẽ gửi một thông báo tới S_2 xin khóa. Nếu dữ liệu này đang bị khóa bởi T_j thì sẽ có 1 cạnh $T_i \rightarrow T_j$ vào LWFG tại vị trí S_2. Nếu trong đồ thị toàn cục mà có chu trình thì *deadlock* cũng xảy ra.



WFG toàn cục.

Khôi phục *deadlock*

Khi xác định rằng hệ thống có *Deadlock* thì chắc chắn rằng hệ thống phải khôi phục do *deadlock*. Cách giải quyết chung là cuộn lại một hay nhiều giao tác. Hai thao tác cơ bản là:

- Chọn giao tác bị nạn: cho 1 tập các giao tác bị *deadlock* ta phải xác định được cần phải cuộn lại một hay nhiều giao tác để giải quyết *deadlock*. Ta cần cuộn lại các giao tác sao cho chi phí là thấp nhất. Các yếu tố xác định chi phí thấp nhất phụ thuộc vào:

+ Giao tác đã được tính toán bao lâu và bao lâu nữa thì hoàn tất.

+ Giao tác đang chiếm giữ bao nhiêu đơn vị dữ liệu.

+ Giao tác cần thêm bao nhiêu đơn vị dữ liệu nưa thì có thể hoàn tất được.

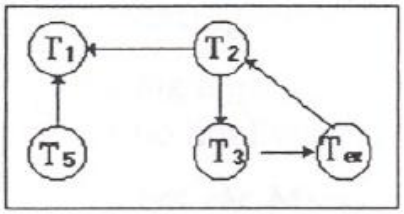
+ Sẽ gồm có bao nhiêu giao tác phải cuộn lại.

- Cuộn giao tác: Khi biết được giao tác nào phải cuộn lại thì xác định tiếp theo là giao tác cuộn lại đến mức nào

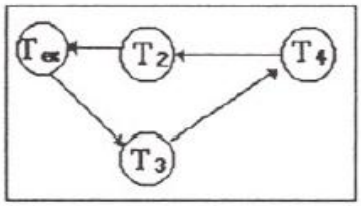
Tiếp cận hoàn toàn phân tán.

Tại một vị trí chứa một đồ thị WFG cục bộ của nó tuy nhiên trong đồ thị này thêm một node gọi là T_{ex} biểu diễn các giao tác bên ngoài vị trí này.

Ví dụ trên sẽ là:



Vị trí 1



Vị trí 2.

Thuật toán dò tìm deadlock phân tán: Giả sử rằng tại vị trí S_i có đồ thị với chu trình của nó chứa node T_{ex}:

$$T_{ex} \longrightarrow T_{k1} \longrightarrow T_{k2} \longrightarrow \cdots \quad T_{kn} \longrightarrow T_{ex}$$

T_{kn} đang đợi để chiếm giữ đơn vị dữ liệu từ vị trí S_j . Sau khi phát hiện ra chu trình thì vị trí S_i sẽ gửi 1 thông báo đến vị trí S_j dò tìm deadlock trong đó có chứa thông tin về chu trình này. Khi đó S_j sẽ cập nhật đồ thị của nó tìm kiếm xem có tồn tại chu trình hay không. Nếu có thì deadlock xảy ra và thực hiện khôi phục deadlock. Nếu chu trình có chứa node T_{ex} thì S_j lại truyền thông báo đến một vị trí S_k thích hợp khác và tiếp tục dò tìm deadlock như trên. Nếu không có thì dừng dò tìm deadlock.

3.2.5 Khôi phục hệ thống với sự điều khiển tương tranh.

Trong hệ thống có nhiều lý do xảy ra hỏng hóc chẳng hạn: đĩa hỏng, mất điện, phần mềm lỗi, không liên lạc được trên đường truyền… tất cả đều có thể làm mất mát dữ liệu. Mỗi một hỏng hóc có thể có nhiều cách giải quyết khác nhau tuy nhiên sau khi khôi phục thì các giao tác đang tương tranh phải tiếp tục như thế nào để không được xảy ra tình trạng mất mát dữ liệu hoặc dữ liệu không nhất quán.

Yêu cầu là nếu có hỏng hóc thì không có đơn vị dữ liệu nào được cập nhật. Vì vậy truớc khi dữ liệu được cập nhật thì phải bảo đảm rằng dữ liệu này đã được bàn giao.

3.2.5.1 Khôi phục dựa trên sổ nhật ký thao tác

Ghi lại tất cả các cập nhật trong một nhật ký thao tác gọi là *log*. Log ghi lại lần lượt các bản ghi chứa các công việc cập nhật trong cơ sở dữ liệu. Một bản ghi gồm:

- Chỉ mục của giao tác (Transaction ID): Cho biết giao tác nào thực hiện thao tác cập nhật.

- Chỉ mục của đơn vị dữ liệu (Data Inhãn ID): Đơn vị dữ liệu nào được cập nhật.

- Giá trị trước cập nhật.

- Giá trị sau cập nhật.

Hoặc là log sẽ khi lại các sự kiện trong quá trình xử lý các thao tác như là trạng thái bắt đầu, commit hoặc abort của giao tác. Khi đó mỗi bản ghi gồm:

- <T_i , start>: giao tác T_i bắt đầu.

- <T_i , X_j , V_1 , V_2>: giao tác T_i thực hiện thao tác ghi trên đơn vị dữ liệu X_j giá trị trước và sau là V_1 và V_2 .

- <T_i commit>: giao tác T_i đã bàn giao.

- <T_i abort>: giao tác T_i đã khôi pục lại trạng thái ban đầu.

Trước khi cập nhật thì ta sẽ ghi vào *log* các bản ghi mô tả quá trình cập nhật.

Dựa trên các điểm kiểm tra.

Khi một hệ thống hỏng hóc xảy ra hệ thống sẽ tra cứu đến *log* của nó để xác định giao tác nào cần thiết phải quay lại trạng thái trước (undo). Có thể duyệt lại toàn bộ *log* để xác định, nhưng để giảm bớt chi phí không cần thiết hệ thống sẽ sử dụng checkpoint.

Thêm vào *log* bản ghi checkpoint. Nếu giao tác T_i bàn giao trước khi checkpoint thì mọi cập nhật của T_i đã được ghi vào cơ sở dữ liệu trước checkpoint . Vì vậy tại thời điểm khôi phục không cần thiết phải thực hiện redo đối với T_i .

Vì vậy nếu có hỏng hóc xảy ra chúng ta cần lần ngược lại để xác định xem trong log thời điểm checkpoint gần nhất .

3.2.5.2 Khôi phục dữ liệu khi thực hiện tương tranh.

Khi cuộn lại một giao tác hỏng nào đó ta sử dụng sổ nhật ký thao tác. Với mỗi bản ghi log có dang: <T_i , X_j , V_1 , V_2> thì đơn vị dữ liệu X sẽ được hồi phục lại giá trị cũ trước đó là V_1 . Cứ như vậy cho đến khi tìm thấy bản ghi <T_i , start> thì kết thúc.

Nếu sử dụng nghi thức khóa 2 pha để điều khiển tương tranh thì việc khóa một đơn vị dữ liệu sẽ kết thúc chỉ khi đã bàn giao hoặc cuộn lại. Vì vậy nếu giao tác nào đó có cuộn lại thì không có giao tác giao tác nào khác cùng cập nhật đơn vị dữ liệu này, do đó việc khôi phục lại giá trị cũ sẽ không ảnh hưởng đến giao tác khác.

Trong hệ thống xử lý các giao tác tương tranh chúng ta đòi hỏi trong *log* phải có bản ghi checkpoint trong đó *log* có dạng <checkpoint L> với L là danh sách các giao tác đang thực hiện tại thời điểm checkpoint.

Thuật toán khôi phục:

a. Bắt dầu khôi phục thì hệ thống cần cấu trúc 2 danh sách:

- Danh sách Undo-list: gồm các giao tác sẽ hủy bỏ.
- Danh sách Redo-list: gồm các giao tác sẽ cho khoiử động lại.

Đầu tiên 2 danh sách này được gán bằng rỗng. Cúnh ta sẽ lần ngược xem xét mỗi bản ghi của log cho đến khi gặp <checkpoint> đầu tiên:

+ Nếu gặp bản ghi <T_i Commit > thì thêm T_i vào danh sách Redo-list.

+ Nếu gặp bản ghi <T_i Start> và nếu Ti không thuộc Redo-list thì thêm vào danh sách Undo-list.

Sau khi duyệt tất cả các bản ghi. Với mỗi T_i trong L nếu T_i không thuộc vào danh sách Redo-list thì thêm T_i vào danh sách Undo-list.

b. Sau khi đã tạo ra 2 danh sách này tiếp tục khôi phục như sau:

- Duyệt lần lượt các bản ghi trong log và thực hiện thao tác hủy bỏ với mỗi bản ghi của giao tác T_i trong Undo-list. Việc duyệt này ngừng lại hoàn toàn khi gặp bản ghi <T_i , Start>.

Các bản ghi trong log của giao tác trong redo-list được bỏ qua trong pha này.

- Định vị lại toàn bộ bản ghi <Checkpoint L> trong log.
- Duyệt lại theo chiều xuôi các bản ghi trong log và thực hiện thao tác khởi động lại với mỗi bản ghi của giao tác T_i trong Redo-list.

Các bản ghi trong log của giao tác trong Undo-list bỏ qua trong pha này.

Trong thuật toán này ta duyệt ngược trong log để thực hiện thao tác Undo cho các giao tác trong Undo-list. Sau đó duyệt xuôi để thực hiện thao tác Redo trong Redo-list. Việc thực hiện này phải đúng thứ tự nếu không sẽ bẫn đến sai.

3.3 CÁC PHƯƠNG PHÁP TRUY CẬP DỮ LIỆU TRONG HỆ PHÂN TÁN.

3.3.1 Các giao tác phân tán.

Mỗi vị trí có chứa 2 hệ thống con:

- TM (transaction manager): Quản lý việc thực hiện các giao tác mà nó truy xuất dữ liệu đến dữ liệu tại vị trí này. Các giao tác có thể chỉ truy xuất dữ liệu tại 1 vị trí hoặc nhiều vị trí. Mỗi TM thực hiện nhiệm vụ như sau:

+ Duy trì 1 log (nhật ký giao tác) cho việc khôi phục dữ liệu.

+ Tham gia vào sơ đồ điều khiển tương tranh thích hợp để điều phối thực hiện tương tranh của các giao tác thực hiện tại vị trí này.

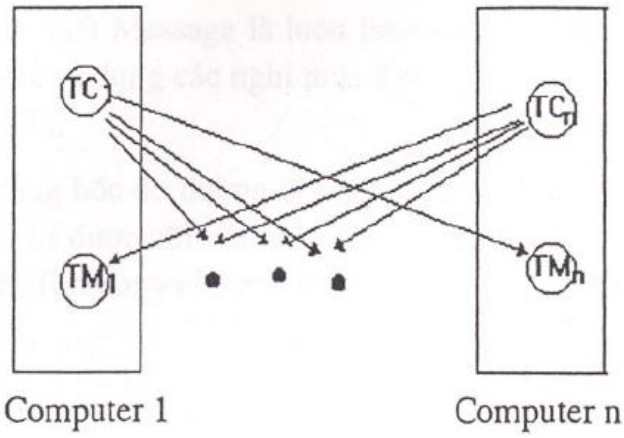
- TC (transaction coordinator): điều phối việc thực hiện của các giao tác mà được khởi tạo tại vị trí này. Với mỗi giao tác TC thực hiện nhiệm vụ sau:

+ Việc bắt đầu thực hiện của giao tác.

+ Chia giao tác toàn cục thành các giao tác giao tác cục bộ và phân phối chúng thực hiện tại các vị trí thích hợp.

+ Việc điều phối kết thúc khi giao tác đã được bàn giao hoặc khôi phục lại tất cả các vị trí.

Nếu bảo đảm tính chất nguyên tử, tất cả các vị trí mà giao tác T thực hiện phải được có chung kết quả cuối cùng. Điều đó có nghĩa là T hoặc là bàn giao tất cả, hoặc là khôi phục tất cả các vị trí. Để đảm bảo tính chất này bộ điều phối giao tác T phải thực hiện một nghi thức thỏa thuận.



Cấu trúc hệ thống của TC và TM

Một nghi thức đơn giản và được sử dụng rộng rãi nhất là nghi thức truyền giao 2 pha, ngoài ra còn có nghi thức truyền giao 3 pha là sự khắc phục nhược điểm của nghi thức truyền giao 2 pha tuy nhiên phức tạp hơn nghi thức 2 pha.

3.3.2 Nghi thức truyền giao 2PC (2 Phase Commit).

Có T là giao tác khởi tạo tại S_i và bộ điều phối giao tác TC tại S_i là C_i . Khi T hoàn tất thực hiện các lệnh của nó, nghĩa là tất cả các vị trí mà T có thực hiện báo cho C_i biết đã hoàn tất, khi đó C_i bắt đầu nghi thức 2PC.

- Pha 1: C_i thêm vào bản ghi <Prepare T> vào log (sổ nhật ký giao tác). Sau đó C_i gửi thông điệp "Prepare T" cho tất cả các vị trí thành viên mà T thực hiện. Sau khi nhận được thông điệp này, các TM tại các vị trí thành viên xác định:

+ Nếu không commit: thêm bản ghi <Abort T> vào log và gửi thông điệp "Vote Abort" cho C_i .

+ Nếu commit: thêm bản ghi <Ready T> vào log và gửi thông điệp "Vote Commit" cho C_i .

- Pha 2: Khi C_i nhận được các câu trả lời từ các vị trí hoặc là thời gian chờ đợi giữa các vị trí vượt quá thừoi gian quy định thì C_i quyết định T là bàn giao hay khôi phục lại.

+ T commit nếu: C_i nhận được các thông điệp "Vote Commit".

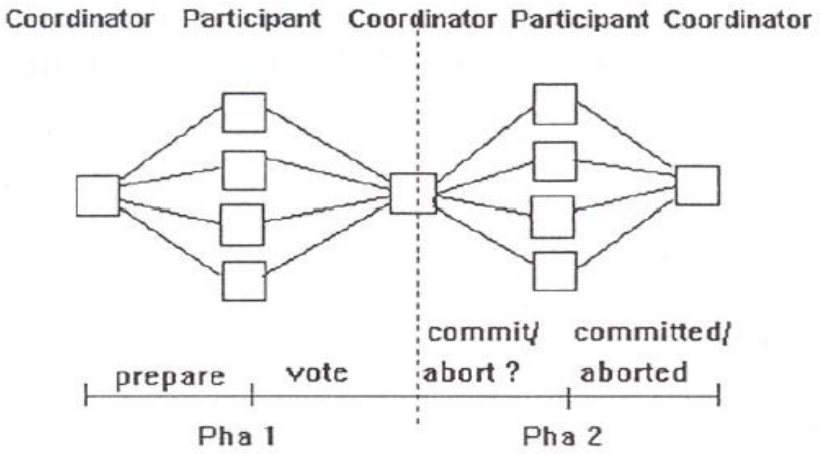
+ T abort nếu: C_i nhận được các thông điệp "Vote Abort".

Và sau đó C_i sẽ thêm bản ghi <Commit T> hoặc <Abort T> vào log sau đó gửi thông điệp "Global Commit" hoặc "Global Abort" cho tất cả các vị trí thành viên. Mỗi vị trí này nhận được thông điệp sẽ ghi vào log của minh và gửi thông điệp phản hồi ACK cho C_i là đã biết quyết định cuối cùng về T và tiến hành bàn giao hặc hủy bỏ tương ứng. Sau khi C_i nhận tát cả các và thông điệp ACK thì thêm vào log bản ghi <Complete T>

Tùy theo vào cấu hình mạng mà ta có các kiểu truyền giao theo nghi thức 2 pha sau.

3.3.2.1. Truyền giao 2PC trung tâm.

Nghi thức này chỉ có sự truyền thông giữa bộ điều phối và các vị trí thành viên, còn các vị trí thành viên thì không có đường truyền giữa chúng. Ta có cấu trúc truyền thông như sau:

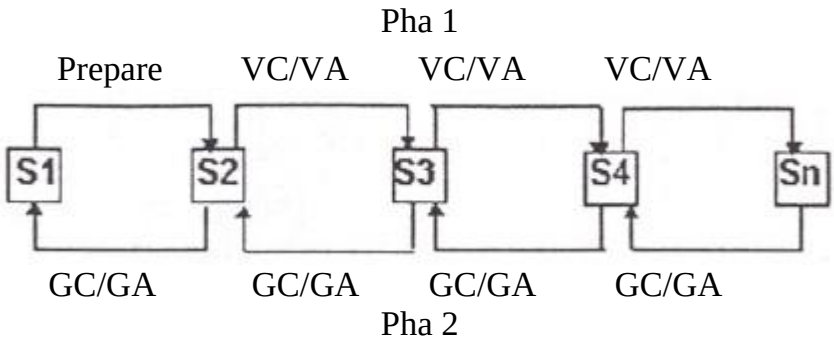


Cấu trúc truyền thông 2PC trung tâm.

3.3.2.2 Truyền giao 2PC tuyến tính.

Nếu mạng hình Bus ta dùng cách truyền giao tuyết tính: giả sử thứ tự của các vị trí là 1,2,…,N. Trong đó bộ diều phối là vị trí 1. Ta sẽ truyền từ vị trí 1 đến vị trí N trong pha 1, pha 2 sẽ truyền ngược lại.

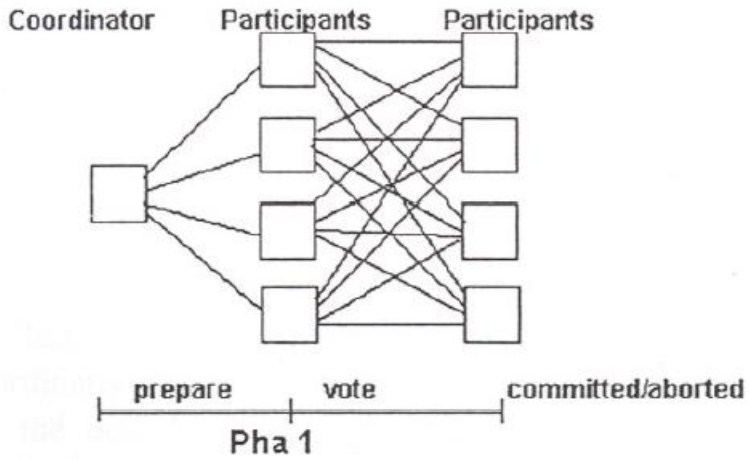
Trước hết bộ điều phối gửi thông điệp <Prepare T> cho vị trí 2, nếu vị trí 2 trả lời chưa sẵn sàng thì vị trí 2 sẽ gửi thông điệp "Vote Abort" (VA) cho vị trí 3, vị trí 2 tự động Abort. Ngược lại cị trí 2 gửi "Vote Commit" (VC) cho vị trí 3. Tiếp tục như vậy đến vị trí N. Kết thúc pha 1. Nếu vị trí N quyết định bàn giao thì gửi N-1 thông điệp "Global Commit T" (GC), ngược lại gửi thông điệp "Global Abort T" (GA).



Cấu trúc truyền thông 2PC tuyến tính.

3.3.2.3 Truyền giao 2PC phân tán.

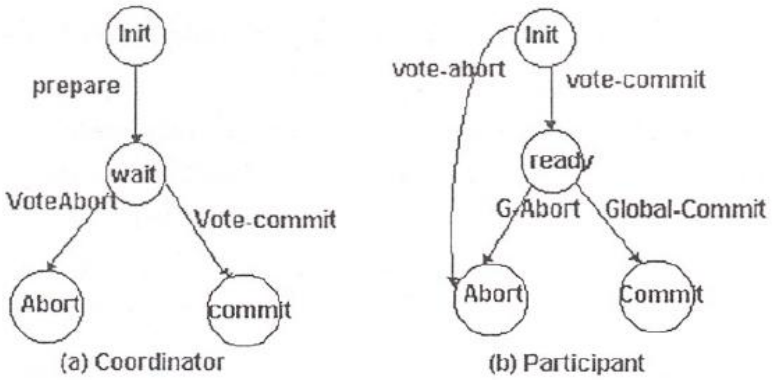
Nghi thức này tất cả các vị trí thành viên đều có nối với nhau, khi đó trong pha 1 tất cả độc lập quyết định là abort hay commit. Bộ điều phối sẽ gửi thông điệp cho tất cả các vị trí thành viên khác. Mỗivị trí thành viên gửi quyết định Yes/No của mình cho tất cả các vị trí khác và cả vị trí điều phối. Mỗi vị trí căn cứ vào các quyết định của các vị trí khác mà quyết định là bàn giao hay khôi pục lại trạng thái ban đầu. Trong nghi thức này có thể không cần 2 pha.



Cấu trúc truyền thông 2PC phân tán.

3.3.2.2 Trường hợp timeout.

Trong nghi thức này, bộ điều phối có các trạng thái sau: Init, Wait, Abort, Commit. Các vị trí thành viên có các trạng thái sau: Init, Ready, Abort, Commit.



Trạng thái giao tác trong nghi thức 2PC

Giả sử trong khi thực hiện nghi thức truyền giao 2PC ở trên thì xảy ra hiện tượng timeout.

- Bộ điều phối timeout: 3 trạng thái có thể timeout: Wait, Abort, Commit.

+ Wait: bộ điều phối đang đợi trả lời từ các vị trí để đưa ra 1 quyết định. Quyết định hủy bỏ và gửi "Global Abort" cho tất cả các vị trí.

+ Abort hay Commit: khi ở trạng thái này bộ điều phối không biết thủ tục bàn giao hoặc hủy bỏ đã được hoàn tất ở vị trí thành viên chưa. Tiếp tục gửi "Global Commit" hoặc "Global Abort" và đợi ACK.

- Vị trí thành viên timeout: có 2 trạng thái có thể xảy ra là: Init và Ready.

+ Init: vị trí thành viên đang đợi "prepare" của bộ điều phối. Như vậy bộ điều phối bị hỏng trong trạng thái init => vị trí thành viên có thể đơn phương hủy bỏ. Nếu sau đó nhận được thông điệp "prepare" thì có thể giải quyết bằng 1 trong 2 cách sau: đọc trong log tìm bản ghi abort và gửi "vote Abort" hoặc bỏ qua thông điệp sau đó bộ điều phối rơi vào trạng thái timeout – wait.

+ Ready: vị trí thành viên đã gửi "Vote Commit" nhưng chưa nhận được quyết định từ bộ điều phối. Vị trí thành viên không thể tự quyết định bàn giao hay hủy bỏ. Ta gọi là bị Blocked cho đến khi nhận thêm thông tin từ bộ điều phối hoặc vị trí khác.

Trong trường hợp các vị trí có thể liên lạc với nhau: nếu xảy ra timeout có thể liên lạc với vị trí khác để đưa ra quyết định. Giả sử P_i timeout các vị trí P_j khác có thể trả lời :

- P_j trong trạng thái Init: P_j chưa gửi quyết định hoặc là P_j chưa nhận được thông điệp "prepare" => P_j trả lời cho P_i là "Vote Abort".

- P_j trong trạng thái Ready: P_j đã gửi "Vote Commit" và đang đợi quyết định toàn cục => P_j không giúp được P_i .

- P_j trong trạng thái Abort hoặc Commit => P_j gửi cho P_i quyết định "Vote Abort " hoặc "Vote Commit" tương ứng.

Khi đó P_i sẽ thực hiện như sau:

- P_i nhận được "Vote Abort" từ tất cả các P_j => P_i chọn hủy bỏ.

- P_i nhận được một số là "Vote Abort", một số là ready => P_i chọn hủy bỏ vì không xảy ra bàn giao toàn cục.

- P_i nhận được tất cả là ready => P_i vẫn bị blocked. => bộ điều phối có thể bị hỏng.

- P_i nhận được tất cả là "Global Commit" hoặc tất cả là "Global Abort" thì P_i đã nhận được quyết định của bộ điều phối.

- P_i nhận được một số là "Global Commit" hoặc "Global Abort" trong khi một số khác ready => Một số vị trí đã nhận được quyết định của bộ điều phối trong khi một số vẫn đợi.

3.3.2.3 Đảm bảo tính nhất quán của dữ liệu

Giả sử trong quá trình thực hiện nghi thức truyêng giao 2PC có vị trí bị hỏng. Vấn đề làm thế nào đảm bảo tính nhất quán của dữ liệu.

- Vị trí thành viên bị hỏng: Sau khi vị trí S_k được khôi phục hoạt động trở lại ta phải xem xét log của giao tác này đang thực hiện đến đâu thì xảy ra hỏng hóc. Các trường hợp là:

+ log chứa "Commit T" => thực hiện Redo(T).

+ log chứa "Abort T" => thự hiện Undo(T).

+ log chứa "Ready T" phải xác định thêm tình trạng của T. Nếu Bộ điều phối vẫn còn thì quan tâm đến T là Commit hay Abort, nếu là Commit thì Redo(T), ngược lại Undo(T). Nếu bộ điều phối không còn thì gửi thông điệp "Query Status T" cho tất cả các vị trí trong hệ thống Sau khi nhận được thì chắc chắn có ít nhất một vị trí tìm trong log xác định xem T là bàn giao hay hủy bỏ, vị trí này sẽ thông báo để S_k biết.

+ log không chứa các loại Abort, Commit, Ready. Khi đó S_k hỏng trước khi nhận được thông điệp "prepare" => S_k thực hiện undo.

- Bộ điều phối bị hỏng: các vị trí sẽ quyết định việc thực hiện của T. Tuy nhiên có những trường hợp các vị trí thành viên không thể quyết định được do đó các vị trí này phải đợi khôi phục dữ liệu trong trường hợp bộ điều phối bị hỏng. Những trường hợp co thể xảy ra:

+ Có 1 vị trí nào mà trong log của nó chứa "Commit T" thì vị trí này đã nhận được "Global Commit" do đó T phải được bàn giao.

+ Có 1 vị trí nào mà trong log của nó chứa "Abort T" thì vị trí này đã nhận được "Global Abort" do đó T phải được Abort.

+ Nếu có các vị trí nào đó không chứa "Ready T" trong log thì bộ điều phối bị hỏng khi nó chưa Commit T => quyết định Abort T.

+ Tất cả các vị trí đều ghi "Ready T" => chúng không thể quyết định được commit hay abort mà phải đợi đến khi bộ điều phối khôi phục dữ liệu.

3.3.3 Nghi thức truyền giao 3PC.

Gọi T là giao tác khởi tạo tại vị trí S_i và bộ điều phối giao tác tại S_i là C_i. Nghi thức này xây dựng để tránh trường hợp blocked.

- Pha 1: C_i thêm vào bản ghi <Prepare T> vào log. Sau đó C_i gửi thông điệp "Prepare T" cho tất cả các vị trí thành viên mà T thực hiện. Sau khi nhận được thông điệp này, các TM tại các vị trí thành viên xác định:

+ Nếu không commit: thêm bản ghi <Abort T> vào log và gửi thông điệp "Vote Abort" cho C_i .

+ Nếu commit: thêm bản ghi <Ready T> vào log và gửi thông điệp "Vote Commit" cho C_i .

- Pha 2:

+ Nếu C_i nhận được "Abort T" hoặc không nhận đầy đủ trong khoảng thời gian quy định thì C_i quyết định Abort. Khi đó C_i sẽ thêm vào log bản ghi <Abort T> và gửi thông điệp "Global Abort" cho các vị trí thành viên.

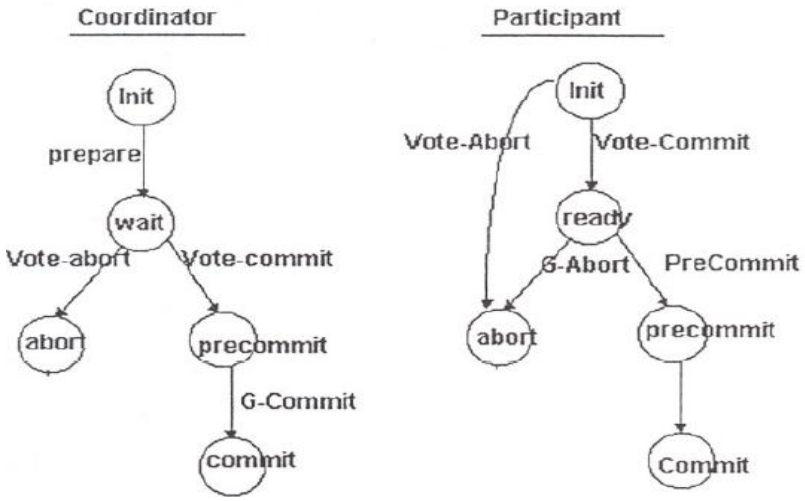
+ Nếu C_i nhận được từ tất cả các thành viên là "Ready T" thì C_i quyết định sơ bộ là precommit. Sau đó nó thêm bản ghi <Precommit T> vào log và gử thông điệp "Precommit T" cho tất cả các thành viên.

Mỗi thành viên nhận được thông điệp từ C_i sẽ ghi vào log của mình và gửi thông điệp ACK cho C_i . Nếu là abort thì tiến hành Abort và sau khi nhận tất cả các thông điệp ACK thì thêm bản ghi <Complete T> vào log. Nếu là Precommit thì tiếp tục pha 3.

- Pha 3: sau khi nhận được tất cả các ACK từ các thành viên thì C_i quyết định commit. C_i thêm bản ghi <Commit T> và gửi thông điệp "Commit T" cho các thành viên. Các thành viên sau khi nhận được thông điệp này sẽ thêm bản ghi <Commit T> và gửi thông điệp ACK cho C_i . Sau khi nhận được tất cả các ACK thì C_i thêm bản ghi <Complete T> vào log.

3.3.2.1 Trường hợp timeout.

Trong nghi thức này, các trạng thái của bộ điều phối là: Init, Wait, Abort, Precommit, Commit. Vị trí thành viên có thể có các trạng thái: Init, Ready, Abort, Precommit, commit.



Các trạng thái của giao tác trong nghi thức truyền giao 3PC.

Giả sử trong khi đang thực hiện nghi thức 3PC thì xảy ra hiện tượng timeout.

- Bộ điều phối timeout: Có 4 trạng thái có thể timeout là: Wait, Precommit, Abort, Commit.

+ Wait: đang đợi các trả lời từ các vị trí để đưa ra quyết định => quyết định Abort và gửi "Global Abort" cho tất cả các vị trí.

+ Precommit: Bộ điều phối biết chắc chắn rằng các thành viên ít nhất là ở trạng thái Ready => tiếp tục thêm bản ghi <commit T> và gửi "Global Commit" cho tất cả các vị trí.

+ Abort hay Commit: Bộ điều phối không biết đã hoàn tất Commit hoặc Abort chưa.

- Vị trí thành viên timeout: các trạng thái có thể xảy ra timeout: Init, Ready, Precommit.

+ Init: đang đợi thông điệp "prepare" của Coordinator => Coordinator bị hỏng trong trạng thái Init => đơn phương Abort.

+ Ready: vị trí thành viên đã gửi "Vote Commit" nhưng chưa nhận được quyết định từ Coordinator => mất liên lạc với bộ điều phối.

+ Precommit: các thành viên đã nhận được "prepare commit".

3.3.2.2 Đảm bảo tính nhất quán dữ liệu.

Khi khôi phục dữ liệu phải đảm bảo tính nhất quán của dữ liệu.

- Vị trí thành viên bị hỏng: Sau khi vị trí thành viên S_k khôi phục trở lại hoặt động bình thường ta phải xem xét lại log của nó để xác định thời điểm xảy ra hỏng hóc.

+ log chứa <Commit T>: thực hiện Redo(T).

+ log chứa <Abort T>: thực hiện Undo(T).

+ log chứa <Ready T>: nếu bộ điều phối vẫn còn thì quan tâm đến T là Commit hay Abort. Nếu Commit thì thực hiện Redo(T), ngược lai Undo(T).

- Vị trí thành viên bị hỏng:

Ta sử dụng thuật toán sau:

+ Các thành viên chọn 1 bộ điều phối mới.

+ Bộ điều phối mới (C_{New}) gửi thông điệp đến các thành viên yêu cầu trả lời về trạng thái của T đang thục hiện tại mỗi vị trí.

+ Mỗi thành viên xác định trạng thái T tại vị trí của mình căn cứ vào log của mình và gửi cho C_{New}:

- Commit: nếu log chứa <Commit T>
- Abort: nếu log chứa <Abort T>
- Ready: nếu log chứa một bản ghi <Ready T>
- Precommit: nếu log chứa một bản ghi <Precommit>
- Not Ready: nếu log không chứa một <Ready T> cũng không chứa một <Abort T> nào.

+ Dựa vào các trả lời cảu các thành viên mà C_{New} quyết định Commit hoặc là Abort hoặc là Restart 3PC như sau:

- Nếu có ít nhất một vị trí Commit =>Commit.
- Nếu có ít nhất một vị trí Abort => Abort.
- Nếu không có vị trí nào Commit và không có vị trí nào Abort và có ít nhất 1 vị trí Precommit => tiếp tục nghi thức 3PC và gửi thông điệp "Precommit" cho các thành viên.
- Còn lại => Abort T.

Thuật toán chọn bộ điều phối mới C_{New} .

Giả sử mỗi thành viên có một số hiêu duy nhất xác định vị trí này, giả sử vị trí S_i là i và giả sử bộ điều phối có số hiệu lớn nhất. Khi bộ điều phối bị hỏng sẽ tìm ra vị trí mới làm C_{New} đồng thời giúp các vị trí thành viên nhận ra đâu là bộ điều phối hiện hành.

Vị trí thành viên S_i gửi thông điệp cho bộ điều phối nhưng vượt qua thời gian t mà không thấy trả lời => bộ điều phối đã bị hỏng => S_i phải tìm C_{New} . S_i sẽ gửi thông điệp "Chọn C_{New} " cho các vị trí có số hiệu lớn hơn. Nó đợi các vị trí này trả lời trong khoảng thời gian t

- Nếu không nhận được trả lời nào => S_i gửi thông báo đến các vị trí có số hiệu nhỏ hơn thông báo nó là C_{New} .

- Nếu S_i có nhận được trả lời trong khoảng thời gian t', thông báo rằng có 1 vị trí có số hiệu cao hơn đã chọn là C_{New} . Nếu quá thời gian t' mà S_i chưa nhận tiếp tục các thông điệp khác từ C_{New} này chứng tỏ vị trí có số hiêu cao hơn này đã bị hỏng và S_i lại phải Restart lại thuật toán này.

3.4 Đánh giá hiệu quả của các phương pháp điều khiển tương tranh.

3.4.1 Ưu khuyết điểm của các phương pháp.

1. Phương pháp dựa trên cơ sở khóa:
   - Phương pháp khóa 2 pha:
   + Ưu điểm: Lịch được tạo ra là khả tuần tự.
   + Khuyết điểm: Có thể xảy ra tình trạng *deadlock*.
   Có thể xảy ra tình trạng hủy bỏ hàng loạt.
   - Phương pháp khóa 2 pha nghiêm ngặt:
   + Ưu điểm: Lịch khả tuần tự.
   Tránh xảy ra hủy bỏ hàng loạt.
   + Khuyết điểm: Có thể xảy ra tình trạng *deadlock*.
   Dữ liệu phải chiếm giữ đến cuối.
2. Phương pháp dựa trên cơ sở nhãn thời gian:
   + Ưu điểm: Lịch được tạo là khả tuần tự đụng độ.
   Tránh *deadlock*.
   + Khuyết điểm : Phải Restart lại nhiều lần.

3.4.2 Các đặc điểm của các phương pháp.

Ta đã xét 2 phương pháp điều khiển tương tranh: phương pháp dựa trên cở sở khóa và phương pháp dựa trên nhãn thời gian. Ta rút ra các đặc điểm các phương pháp như sau:

1. Có 2 cách cơ bản để cho thực hiện tương tranh: waiting (đợi) hoặc restart (khởi động lại) các giao tác. Phương pháp khóa dùng waiting, phương pháp nhãn thời gian dùng restarting.

2. Thứ tự nối tiếp của các giao tác được xác định bằng 2 cách sau: hoặc bằng thứ tự truy xuất các phần tử dữ liệu hoặc bằng cách gán nhãn thời gian cho các giao tác.

3. Với tất cả các phương pháp tạo ra việc chờ đợi, chúng ta cần phải có giải pháp cho vấn đề *deadlock*. Giải pháp này có thể là phát hiện hay ngăn ngừa *deadlock*. Trong đó việc giải quyết *deadlock* luôn yêu cầu việc khởi động lại một giao tác.

KẾT LUẬN

Đồ án đã tìm hiểu, giới thiệu các phương pháp điều khiển tương tranh trong hệ phân tán đó là: phương pháp dựa trên cơ sở khóa và phương pháp dựa trên nhãn thời gian. Phương pháp dựa trên cơ sở khóa thì thứ tự mỗi cặp đụng độ được xác định tại thời điểm khóa đầu tiên còn thuật toán dựa trên nhãn thời gian chọn lựa thứ tự theo cách là giao tác nào đến trước. Mỗi phương pháp đều có ưu và khuyết điểm riêng. Đồng thời giới thiệu cách khôi phục lại dữ liệu khi xảy ra sự tương tranh dẫn đến hỏng hóc mà dẫn đến giao tác phải hủy bỏ. Giới thiệu hai giao thức truyền giao là nghi thức truyền giao 2 pha và nghi thức truyền giao 3 pha (là nghi thức khắc phục nhược điểm của nghi thức 2 pha nhưng phức tạp hơn nhiều).

Đồ án còn nhiều thiếu sót kính mong nhận được sự góp ý của thầy cô và các bạn. Em hi vọng những kết quả đạt được sẽ có ích cho việc xây dựng hệ phân tán đáp ứng nhu cầu ngày càng cao hiện nay và đảm bảo tính an toàn, toàn vẹn dữ liệu.

TÀI LIỆU THAM KHẢO

[1]. Thạc Bình Cường, *Phân tích thiết kế hệ thống thông tin*, NXB Thống Kê HN.

[2]. Thạc Bình Cường, *Giáo trình cơ sở dữ liệu: lý thuyết và thực hành,* NXB Bưu điện Hà Nội, 2004

[3]. *Giáo trình cơ sỏ dữ liệu phân tán*. Khoa CNTT, ĐHKH TN, ĐHQG Hà Nội.

[4].Nguyễn Văn Định, *Thuật toán tìm kết nối tối ưu cho cơ sở dũ liệu phân tán.* Báo cáo tại đại hội CNTT toàn Quốc, Khê Lải, 2007.

[6]. Phạm Thị Anh Lê , *Cơ sở dữ liệu phân tán.* NXB Bưu Điện HN.

[7]. Nguyễn Tuệ, *Nhập môn cơ sở dữ liệu.* NXB Giáo Dục.

[8]. Đỗ Phúc, *Chuyên đề cơ sở dữ liệu nâng cao*.

[9].Chu Kỳ Quang, Phạm Văn Cừ, *Tiêu chuẩn khả tuần tự của các giao dịch tương tranh sử dụng ngữ nghĩa các phép toán,*