

Chương 6: ĐIỀU KHIỂN ĐỒNG THỜI PHÂN TÁN.

MỤC TIÊU:

Như đã thảo luận trong chương 5, điều khiển đồng thời giải quyết các tính chất *biệt lập* (isolation) và *nhất quán* (consistency) của giao dịch. Cơ chế điều khiển đồng thời phân tán của một hệ quản trị CSDL phân tán bảo đảm rằng tính nhất quán của CSDL, như đã được định nghĩa trong phần 5.2.2.

Trong chương này chúng ta đưa ra một giả thiết quan trọng: hệ thống phân tán hoàn toàn khả tín và không có bất kỳ sự cố nào (cả phần cứng lẫn phần mềm).

6.1 LÝ THUYẾT KHẢ TUẦN TỰ

Trong phần 5.3 chúng ta thảo luận vấn đề làm biệt lập các giao tác với nhau theo tác dụng của chúng trên CSDL. Chúng ta cũng đã chỉ ra rằng nếu việc thực thi đồng thời các giao tác làm cho CSDL ở một trạng thái có thể có được giống như khi cho chúng thực hiện tuần tự theo một số thứ tự nào đó, các vấn đề như cập nhật bị thất lạc sẽ được giải quyết. Đây là điểm mấu chốt của những lý luận về tính khả tuần tự. Phần còn lại sẽ tập trung vào các vấn đề khả tuần tự một cách hình thức hơn.

Một *lịch* S (schedule) được định nghĩa trên tập giao tác $T = \{T_1, T_2, \dots, T_n\}$ và xác định thứ tự thực thi đan xen lẫn nhau của các thao tác trong giao dịch. Dựa trên định nghĩa giao tác đã được giới thiệu trong phần 5.1, lịch có thể mô tả như một thứ tự bộ phận trên T . Đầu vậy chúng ta cũng cần một khái niệm cơ bản trước khi đưa ra định nghĩa hình thức.

Nhắc lại định nghĩa về thao tác tương tranh đã đưa ra trong chương 5. Hai thao tác $O_{ij}(x)$ và $O_{kl}(x)$ (i và k không nhất thiết phải phân biệt) cùng truy cập đến một thực thể CSDL x được gọi là *có tương tranh* (conflict) nếu ít nhất một trong chúng là *thao tác ghi* (write). Hai điều mà chúng ta cần chú ý trong định nghĩa này:

- Trước tiên các thao tác đọc không tương tranh với nhau. Vì thế chúng ta có thể nói về hai loại tương tranh: *đọc-ghi* (read-write) và *ghi-ghi* (write-write).
- Thứ hai, hai thao tác này có thể thuộc về cùng một giao tác hoặc thuộc về hai giao tác khác nhau. Trong trường hợp sau, hai giao tác được gọi là có tương tranh. Về trực quan, sự tồn tại của một tương tranh giữa hai thao tác cho thấy rằng thứ tự thực hiện của chúng là quan trọng. Việc sắp thứ tự cho hai thao tác đọc là không cần thiết.

Trước tiên chúng ta định nghĩa *một lịch đầy đủ* (complete schedule): là lịch định nghĩa thứ tự thực hiện của tất cả các thao tác trong miền biến thiên của nó. Sau đó chúng ta định nghĩa rằng một lịch được xem là một *tiền tố* (prefix) của một lịch đầy đủ. Về hình thức, một lịch đầy đủ S_T^c được định nghĩa trên một tập giao tác $T = \{T_1, T_2, \dots, T_n\}$ là một thứ tự bộ phận $= \{\Sigma_T, <_T\}$, trong đó

1. $\Sigma_T = \bigcup_{i=1}^n \Sigma_i$
2. $<_T = \bigcup_{i=1}^n <_i$
3. Đối với hai thao tác trong tương tranh bất kỳ $O_{ij}, O_{kl} \in \Sigma_T$, chúng ta có $O_{ij} <_T O_{kl}$ hoặc $O_{kl} <_T O_{ij}$.

Thí dụ 1

T_1 :	Read(x)	T_2 :	Read(x)
	$X \leftarrow x + 1$		$X \leftarrow x + 1$
	Write(x)		Write(x)
	Commit		Commit

Một lịch đầy đủ S_T^c khả hữu trên $T = \{T_1, T_2\}$ có thể được viết như thứ tự bộ phận sau đây (các chỉ số dưới biểu thị giao dịch):

$$S_T^c = \{\Sigma_T, <_T\}$$

trong đó

$$\Sigma_1 = \{R_1(x), W_1(x), C_1\}$$

$$\Sigma_2 = \{R_2(x), W_2(x), C_2\}$$

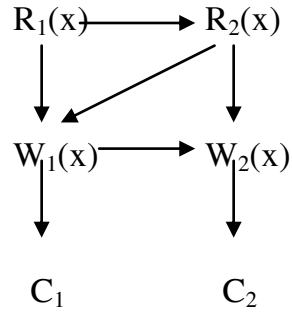
Vì vậy

$$\Sigma_T = \Sigma_1 \cup \Sigma_2 = \{R_1(x), W_1(x), C_1, R_2(x), W_2(x), C_2\}$$

và

$$<_T = \{(R_1, R_2), (R_1, W_1), (R_1, C_1), (R_1, W_2), (R_1, C_2), (R_2, W_1), (R_2, C_1), (R_2, W_2), (R_2, C_2), (W_1, C_1), (W_1, W_2), (W_1, C_2), (C_1, W_2), (C_1, C_2), (W_2, C_2)\}$$

có thể đặc tả như một DAG trong Hình 6.1.



Hình 6.1. Biểu diễn DAG của một lịch đầy đủ.

$$S_T^c = \{R_1(x), R_2(x), W_1(x), C_1, W_2(x), C_2\}$$

Một lịch được định nghĩa là một *tiền tố* (prefix) của một lịch đầy đủ. Một tiền tố của một thứ tự bộ phận có thể được định nghĩa như sau. Cho trước một thứ tự bộ phận $P = \{\Sigma, <\}$, $P' = \{\Sigma', <'\}$ là một tiền tố của P nếu

1. $\Sigma' \subseteq \Sigma$;
2. $\forall e_i \in \Sigma', e_i <' e_j$ nếu và chỉ nếu $e_i < e_j$; và
3. $\forall e_i \in \Sigma'$, nếu $\exists e_j \in \Sigma$ và $e_j < e_i$ thì $e_j \in \Sigma'$.

Hai điều kiện đầu tiên định nghĩa P' như một hạn chế của P trên miền Σ' , trong đó các quan hệ thứ tự trong P được duy trì trong P' . Điều kiện cuối cùng chỉ ra rằng với mọi phần tử của Σ' , tất cả các phần tử đứng trước nó trong Σ cũng phải thuộc Σ' .

Định nghĩa một lịch thức như một tiền tố của một thứ tự bộ phận để làm gì? Câu trả lời đơn giản là chúng ta bây giờ có thể xử lý các lịch không đầy đủ. Điều này là có ích vì một số lý do. Từ quan điểm lý thuyết khả tuần tự, chúng ta chỉ phải giải quyết một số thao tác của các giao tác có tương tranh chứ không phải với tất cả mọi thao tác. Hơn nữa, và có lẽ quan trọng hơn là khi xuất hiện sự cố, chúng ta cần phải có khả năng giải quyết với những giao tác không đầy đủ, mà đó là điều một tiền tố cho phép chúng ta làm được.

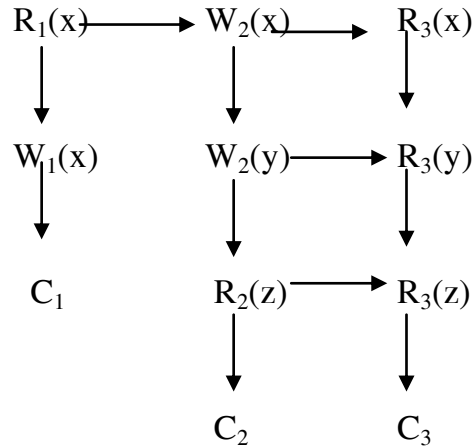
Thí dụ 2

Xét ba giao tác sau đây

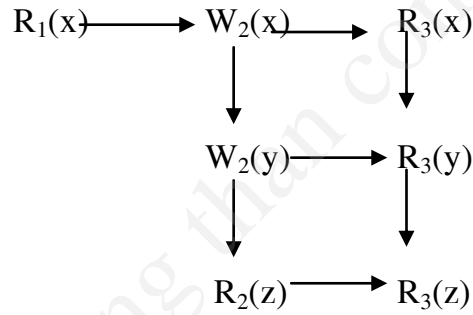
T ₁ :	Read(x)	T ₂ :	Write(x)	T ₃ :	Read(x)
	Write(x)		Write(y)		Read(y)
	Commit		Read(z)		Read(z)
			Commit		Commit

Chương 6: Điều khiển đồng thời phân tán

Một lịch đầy đủ S^c cho những giao tác này được trình bày trong hình 6.2, và một lịch S (một tiền tố của S^c) được mô tả trong hình 6. 3.



Hình 6.2. Một lịch đầy đủ.



Hình 6.3. Tiền tố của lịch đầy đủ của hình 6.2.

Nếu trong lịch S , các thao tác của các giao tác khác nhau không được thực hiện xen kẽ (nghĩa là các thao tác của mỗi giao tác xảy ra liên tiếp), lịch được gọi là *tuần tự* (serial).

Thí dụ 3

Xét ba giao tác của thí dụ 2. Lịch sau đây:

$$S = \{W_2(x), W_2(y), R_2(z), C_2, R_1(x), W_1(x), C_1, R_3(x), R_3(y), R_3(z), C_3\}$$

là tuần tự bởi vì tất cả các thao tác của T_2 được thực hiện trước tất cả các thao tác của T_1 và tất cả thao tác của T_1 được thực hiện trước tất cả các thao tác của T_3 . Một cách thường được dùng để biểu thị mối liên hệ thứ bậc giữa các thực thi giao tác là $T_2 <_s T_1 <_s T_3$ hoặc $T_2 \rightarrow T_1 \rightarrow T_3$.

Về trực quan, hai lịch S_1 và S_2 được định nghĩa trên cùng một tập giao tác T được gọi là *tương đương* nếu với mỗi cặp thao tác tương tranh O_{ij} và O_{kl} ($i \neq k$), mỗi khi $O_{ij} <_1 O_{kl}$ thì $O_{ij} <_2 O_{kl}$. Đây được gọi là *tương đương tương tranh* (conflict equivalence) bởi vì nó định nghĩa sự tương đương của hai lịch theo thực thi tương đối của các thao tác tương tranh trong các lịch biểu.

Thí dụ 4

Chúng ta xét lại ba giao tác của thí dụ 2. Lịch S dưới đây được định nghĩa trên chúng là tương đương tương tranh với S của thí dụ 3:

$$S' = \{W_2(x), R_1(x), W_1(x), C_1, R_3(x), W_2(y), R_3(y), R_2(z), C_2, R_3(z), C_3\}$$

Một lịch S được gọi là *khả tuần tự* (serializable) nếu và chỉ nếu có tương đương tương tranh với một lịch tuần tự.

Chú ý rằng tính khả tuần tự chỉ tương đương với tính nhất quán độ 3 đã được định nghĩa trong phần 5.2.2. Tính khả tuần tự được định nghĩa như thế cũng được gọi là *khả tuần tự theo tương tranh* bởi vì nó được định nghĩa theo sự tương đương tương tranh.

Thí dụ 5

Lịch S' trong thí dụ 4 là khả tuần tự bởi vì nó tương đương với lịch tuần tự S của thí dụ 3. Cũng chú ý rằng vấn đề khi thực hiện một cách không kiểm soát các giao tác T_1 và T_2 của thí dụ 10.8 đó là chúng có thể sinh ra một lịch bất khả tuần tự.

Bây giờ khi đã định nghĩa một cách hình thức tính khả tuần tự, chúng ta có thể chỉ ra rằng chức năng cơ bản của bộ phận điều khiển đồng thời là tạo ra một lịch khả tuần tự để thực hiện các giao tác đang chờ đợi.

Lý thuyết khả tuần tự có thể mở rộng cho các CSDL phân tán không nhân bản (hoặc phân hoạch). Lịch thực thi giao tác tại mỗi vị trí được gọi là *lịch cục bộ* (local schedule). Nếu CSDL không được nhân bản và mỗi lịch cục bộ đều khả tuần tự thì hợp của chúng (được gọi là lịch toàn cục) cũng khả tuần tự, với điều kiện là các thứ tự tuần tự hoá cục bộ đều giống nhau. Tuy nhiên trong các CSDL phân tán có nhân bản, mở rộng lý thuyết khả tuần tự đòi hỏi phải cẩn trọng. Có thể là các lịch cục bộ khả tuần tự nhưng tính nhất quán tương hỗ của CSDL vẫn bị tổn hại.

Thí dụ 6

Chúng ta sẽ đưa ra một thí dụ rất đơn giản nhằm minh họa cho điều này. Xét hai vị trí và một mục dữ liệu (x) hiện diện cả hai tại cả hai nơi. Xét giao tác sau:

T_1 :	Read(x)	T_2 :	Read(x)
	$x \leftarrow x + 5$		$x \leftarrow x + 5$
	Write(x)		Write(x)
	Commit		Commit

Rõ ràng cả hai giao tác đều phải thực hiện ở cả hai nơi. Xét các lịch có thể được tạo ra tại hai vị trí đó:

$$S_1 = \{R_1(x), W_1(x), C_1, R_2(x), W_2(x), C_2\}$$
$$S_2 = \{R_2(x), W_2(x), C_2, R_1(x), W_1(x), C_1\}$$

Tính nhất quán tương hỗ đòi hỏi rằng tất cả các giá trị của mọi mục dữ liệu nhân bản đều phải như nhau. Các lịch có thể duy trì được tính nhất quán tương hỗ được gọi là *khả tuần tự một bản* (one-copy serializable).

Về trực quan, lịch toàn cục khả tuần tự một bản phải thỏa mãn những điều kiện sau:

1. Mỗi lịch cục bộ đều phải khả tuần tự.
2. Hai thao tác tương tranh phải có cùng thứ tự tương đối trong tất cả các lịch cục bộ nơi mà chúng cùng xuất hiện.

Điều kiện thứ hai chỉ nhằm bảo đảm rằng thứ tự tuần tự hóa đều như nhau tại tất cả mọi vị trí có các giao tác tương tranh cùng thực hiện. Cần nhớ rằng các thuật toán điều khiển đồng thời bảo đảm được tính khả tuần tự bằng cách đồng bộ hóa các truy xuất tương tranh đến CSDL. Trong các CSDL nhân bản, nhiệm vụ bảo đảm tính khả tuần tự một bản thường là trách nhiệm của *nghi thức điều khiển bản sao* (replica control protocol).

Chúng ta hãy giả sử là tồn tại một mục dữ liệu x với các bản sao x_1, x_2, \dots, x_n . Chúng ta sẽ xem như x là một mục dữ liệu logic và mỗi bản sao là một mục dữ liệu vật lý. Nếu tính vô hình nhân bản được cung cấp, các giao tác của người sử dụng sẽ đưa ra các thao tác đọc và ghi trên mục dữ liệu x . Nghi thức điều khiển bản sao chịu trách nhiệm ánh xạ mỗi thao tác đọc trên mục dữ liệu logic x [Read(x)] thành thao tác đọc trên một trong những bản sao vật lý x_j của x [Read(x_j)]. Ngược lại, mỗi thao tác ghi trên mục logic x được ánh xạ thành một tập thao tác ghi trên một tập con (có thể là tập con thực sự) của các bản sao vật lý x . Bất kể ánh xạ chuyển

Chương 6: Điều khiển đồng thời phân tán

đến toàn bộ tập của các bản sao hay chỉ đến một tập con thì nó vẫn là cơ sở để phân loại các thuật toán điều khiển bản sao. Trong chương này và phần lớn cuốn sách, chúng ta sẽ xét các nghi thức điều khiển bản sao ánh xạ một thao tác đọc trên một mục logic đến một bản sao của nó nhưng lại ánh xạ thao tác ghi thành tập các thao tác ghi trên tất cả các bản sao vật lý. Nghi thức này thường được gọi là *nghi thức đọc một/ghi tất cả* (read-one/write-all protocol, ROWA).

Một nhược điểm của nghi thức ROWA là nó làm giảm độ khả dụng của CSDL, khi có sự cố bởi vì giao tác có thể không hoàn tất được trừ khi nó đã phản ánh tác dụng của tất cả các thao tác ghi trên các bản sao.

Vì thế có một số thuật toán cố gắng duy trì tính nhất quán tương hồ mà không sử dụng đến nghi thức ROWA. Chúng đều dựa trên một tiền đề là chúng ta vẫn có thể tiếp tục tiến hành một thao tác, miễn là thao tác này có thể được xếp lịch tại một tập con các vị trí chiếm hơn phân nửa các vị trí có lưu các bản sao hoặc là tất cả các vị trí có thể đến được (nghĩa là còn dùng được). Vẫn có những nghi thức khác thực hiện các cập nhật trên một bản chính được chọn ra của mục dữ liệu nhân bản rồi lan truyền các cập nhật này cho những bản sao khác vào thời điểm thích hợp.

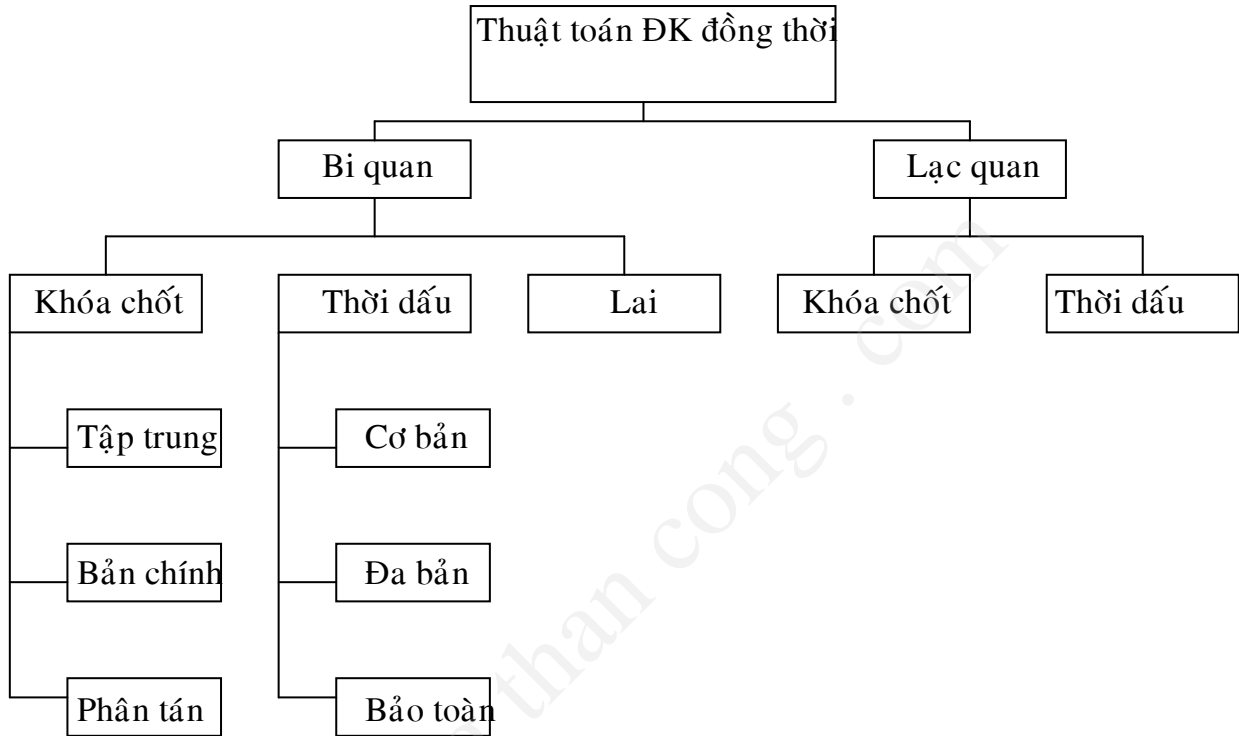
6.2 PHÂN LOẠI CÁC CƠ CHẾ ĐIỀU KHIỂN ĐỒNG THỜI

Có một số cách phân loại các phương pháp điều khiển đồng thời. Một tiêu chuẩn hiển nhiên là chế độ phân tán CSDL. Một số thuật toán đã được đề xuất đòi hỏi có một CSDL nhân bản hoàn toàn, còn một số khác có thể hoạt tác trên các CSDL phân hoạch hoặc nhân bản một phần. Các thuật toán điều khiển đồng thời cũng có thể được phân loại theo topo mạng, chẳng hạn như một mạng con phải có khả năng phát tán hoặc các thuật toán hoạt động trên các mạng hình sao hoặc các mạng kết vòng.

Tuy nhiên tiêu chuẩn phân loại thông dụng nhất là theo *nguyên thủy đồng bộ hóa* (synchronization primitive). Sự phân chia tương ứng đưa các thuật toán điều khiển đồng thời vào hai lớp: những thuật toán dựa trên các truy xuất độc quyền đến dữ liệu dùng chung (khóa chốt) và những thuật toán cố gắng sắp thứ tự hiện giao tác theo một tập qui tắc (nghi thức). Tuy nhiên các nguyên thủy này đều có thể dùng được dùng trong các thuật toán với hai quan điểm khác nhau: *quan điểm bi quan* (pessimistic view) cho rằng có nhiều giao tác sẽ tương tranh với nhau, còn *quan điểm lạc quan* (optimistic view) cho rằng không có quá nhiều giao tác tương tranh với nhau.

Vì vậy chúng ta sẽ xếp các cơ chế điều khiển đồng thời thành hai nhóm lớn: các phương pháp điều khiển đồng thời lạc quan và các phương pháp điều khiển đồng thời bi quan. Các thuật toán bi quan đồng bộ hóa việc thực hiện đồng thời của các

giao tác trước khi thực hiện chúng, trong khi đó các thuật toán lạc quan để việc đồng bộ hóa các giao tác cho đến khi chúng kết thúc. Nhóm lạc quan gồm có các *thuật toán dựa theo khóa chốt* (locking-based algorithm), các thuật toán bi quan dựa theo thứ tự giao tác và các *thuật toán lai* (hybrid algorithm). Tương tự, nhóm lạc quan cũng có thể được phân loại thành các thuật toán dựa theo khóa và các thuật toán theo thứ tự thời gian. Phân loại này được trình bày trong hình 6.4.



Hình 6.4. Phân loại các thuật toán điều khiển đồng thời.

Trong cách tiếp cận dùng khoá chốt, việc đồng bộ hóa giao tác có được bằng cách sử dụng các khoá chốt vật lý hoặc logic trên một phần CSDL. Kích thước của các phần này (thường được gọi là độ *mịn khóa*, locking granularity) là một vấn đề quan trọng. Tuy nhiên trong lúc này chúng ta sẽ bỏ qua nó và xem kích thước được chọn là *một đơn vị khóa* (lock unit). Lớp cơ chế này được chia nhỏ hơn nữa tùy theo vị trí thực hiện các hoạt động quản lý khóa:

1. Trong lối khóa tập quyền, một trong các vị trí của mạng được chỉ định làm vị trí chính, ở đó lưu trữ các bảng khóa cho toàn bộ CSDL và chịu trách nhiệm trao khóa cho các giao dịch.
2. Theo lối khóa bản chính thì ngược lại, một trong các bản sao (nếu có nhiều bản) của mỗi đơn vị khóa được chỉ định làm *bản chính* (primary copy), và đó chính là bản sẽ bị khóa khi giao tác truy xuất đến đơn vị đó. Ví dụ nếu đơn vị khóa x được nhân bản tại các vị trí 1,2, và 3, một trong những vị trí này (chẳng hạn 1) được chọn làm vị trí chính cho x. Tất cả mọi giao tác muốn truy xuất x sẽ nhận được một khóa của chúng tại vị trí

1 trước khi chúng có thể truy xuất được một bản sao của x . Nếu CSDL không được nhân bản (nghĩa là mỗi đơn vị khóa chỉ có một bản duy nhất), các cơ chế khóa bản chính sẽ phân phối trách nhiệm quản lý khóa cho một số vị trí.

3. Theo lối khóa phi tập trung, nhiệm vụ quản lý khóa là của tất cả các vị trí trong mạng. Trong trường hợp này, thực hiện một giao tác có sự tham gia và điều phối của các bộ xếp lịch tại nhiều vị trí. Mỗi bộ xếp lịch cục bộ chịu trách nhiệm về các đơn vị khóa nằm cục bộ tại vị trí đó. Trong thí dụ trên, các thực thể muốn truy xuất x phải nhận được khóa tại tất cả ba vị trí.

Lớp cơ chế theo *thứ tự thời dấu* (timestamp ordering, viết tắt là TO) phải tổ chức thứ tự thực hiện của các giao tác nhằm duy trì được tính nhất quán lẫn tương hỗ giữa các vị trí (liên nhất quán). Việc xếp thứ tự này được duy trì bằng cách gán thời dấu cho cả giao tác lẫn mục dữ liệu được lưu trong CSDL. Những thuật toán này có thể thuộc loại *cơ bản* (basic TO), *đa phiên bản* (multiversion TO), hoặc *bảo toàn* (conservative TO).

Chúng ta cần chỉ ra rằng một số thuật toán dựa theo khóa cũng có thể dùng thời dấu, chủ yếu nhằm cải thiện hiệu quả và mức độ hoạt động đồng thời. Chúng ta gọi lớp thuật toán này là thuật toán lai. Chúng ta sẽ không thảo luận về chúng trong chương này bởi vì chúng chưa hề được cài đặt trong các hệ quản trị CSDL phân tán thương mại và các hệ thử nghiệm.

6.3 CÁC THUẬT TOÁN ĐIỀU KHIỂN ĐỒNG THỜI BẰNG KHÓA CHỐT

Ý tưởng chính của việc điều khiển đồng thời bằng khóa chốt là bảo đảm dữ liệu dùng chung cho các thao tác tương tranh chỉ được truy xuất mỗi lần một giao dịch. Điều này được thực hiện bằng cách liên kết *một khóa chốt* (lock) với mỗi đơn vị khóa. Khóa này được giao tác đặt ra trước khi nó truy xuất và được điều chỉnh lại vào lúc nó hết sử dụng. Hiển nhiên là một đơn vị khóa không thể truy xuất được nếu đã bị khóa bởi một giao tác khác. Vì vậy yêu cầu khóa của một giao tác chỉ được trao nếu khóa đi kèm hiện không bị một giao tác khác giữ.

Bởi vì chúng ta quan tâm đến việc đồng hóa các thao tác tương tranh của các giao tác tương tranh nên có hai loại khóa chốt (thường được gọi là *thể thức khóa*, lock mode) được kèm với mỗi đơn vị khóa: *khóa đọc* (real lock, rl) và *khóa ghi* (write lock, wl). Một giao tác T_i đang muốn đọc một mục dữ liệu được chứa trong đơn vị khóa x sẽ nhận được một khóa đọc trên x [ký hiệu là $rl_i(x)$] và cũng tương tự với các thao tác ghi. Thường thì chúng ta hay nói về *tính tương thích* (compatibility) của các thể thức khóa chốt. Hai thể thức khóa là *tương thích* nếu hai giao tác truy xuất đến cùng một mục dữ liệu có thể nhận được khóa trên mục dữ liệu đó cùng

Chương 6: Điều khiển đồng thời phân tán

một lúc. Như Hình 6.5 cho thấy, các khóa đọc là tương thích với nhau, còn các khóa đọc-ghi hoặc ghi-ghi thì không. Vì vậy hai giao tác vẫn có thể đồng thời đọc cùng một mục.

	$rl_i(x)$	$wl_i(x)$
$rl_j(x)$	tương thích	không tương thích
$wl_j(x)$	không tương thích	không tương thích

Hình 6.5. Ma trận tương thích của các thể thức khóa

Các DBMS phân tán không chỉ phải quản lý các khóa mà còn phải có trách nhiệm xử lý khóa dùng cho giao dịch. Nói cách khác, người sử dụng không cần phải xác định khi nào phải khóa dữ liệu; DBMS phân tán sẽ lo liệu điều đó mỗi khi các giao tác đưa ra yêu cầu đọc hoặc ghi.

Trong các hệ thống dùng khóa chốt, bộ xếp lịch (scheduler) (xem hình 6.4) chính là bộ quản lý khóa (lock manager, LM). Bộ quản lý giao tác sẽ chuyển cho bộ quản lý khóa các thao tác CSDL (đọc hoặc ghi) và các thông tin kèm theo (như mục dữ liệu cần truy xuất, định danh của giao tác đưa ra yêu cầu). Sau đó bộ quản lý khóa sẽ kiểm tra xem đơn vị khóa có chứa mục dữ liệu đó đã bị khóa hay chưa. Nếu đã khóa, và nếu thể thức khóa đó không tương thích với thể thức của giao tác đang yêu cầu, thao tác sẽ bị hoãn lại. Ngược lại, khóa sẽ được đặt với thể thức mong muốn và thao tác này được chuyển cho bộ xử lý dữ liệu để truy xuất CSDL thực sự. Sau đó bộ quản lý giao tác được thông tin về các kết quả thực hiện. Việc kết thúc giao tác sẽ giải phóng các khóa của nó và làm khởi hoạt một giao tác khác đang đợi truy xuất mục dữ liệu này.

Thuật toán khóa chốt cơ bản nằm trong thuật toán 6.1. Hình 6.6 đưa ra các khai báo kiểu và các định nghĩa thủ tục được dùng trong các thuật toán của chương này. Chú ý trong thuật toán 6.1, chúng ta không quan tâm đến cách thực thi các thao tác ủy thác và hủy bỏ giao dịch.

Declare-type

Operation: một trong số Begin-Transaction, Read, Write, Abort, hoặc Commit

DataItem: một mục dữ liệu trong CSDL phân tán

TransactionId: một giá trị duy nhất được gán cho mỗi giao dịch.

DataVal: một giá trị có kiểu dữ liệu cơ bản (nghĩa là số nguyên, số thực, văn bản)

SiteId: một định danh duy nhất cho vị trí

Dbop: một bộ ba gồm { một phép toán trên CSDL của ứng dụng }

opn: Operation

Chương 6: Điều khiển đồng thời phân tán

data: DataItem
tid: TransactionId

Dpmsg: một bộ ba gồm
opn: Operation
tid: TransactionId
result: DataVal

Scmsg: một bộ ba gồm
opn: Operation
tid: TransactionId
result: DataVal

Transaction \leftarrow một bộ hai gồm
tid: TransactionId
body: thân giao tác

Message \leftarrow một chuỗi ký tự cần được truyền đi
OpSet: một tập các Dbop
SiteSet: một tập các SiteId
WAIT(msg: Message)
begin
 {đợi cho đến khi có một thông báo đến}
end

Hình 6.6. Các định nghĩa chuẩn bị cho các thuật toán sắp tới

Thuật toán 6.1. Bộ quản lý khóa cơ bản (Basic LM)

Declare-var

msg: Message
dop: Dbop
Op: Operation
x: DataItem
T: TransactionId
pm: Dpmsg
res: DataVal
SOP: OpSet

Begin

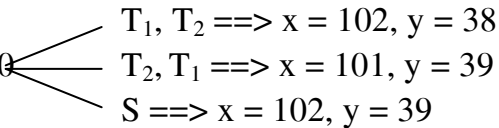
repeat
 WAIT(msg)
 case of msg
 Dbop: // phép toán

```
begin
  Op  $\leftarrow$  dop.opn
  x  $\leftarrow$  dop.data
  T  $\leftarrow$  dop.tid
case of Op
  Begin_Transaction, Abort or Commit:
    begin
      gửi dop đến bộ xử lý dữ liệu
    end
  Read or Write:
    begin
      tìm đơn vị khóa (lock unit) lu sao cho  $x \subseteq lu$ 
      if lu chưa bị khóa or thể thức khóa của lu tương thích với Op then
        begin
          đặt khóa trên lu ở thể thức thích hợp
          gửi dop đến bộ xử lý dữ liệu
        end
      else đưa dop vào một hàng đợi của lu
      end-if
    end
  end-case // Op của Dbop
  Dpmsg: // Thông báo từ bộ xử lý dữ liệu           { trả lời của bộ xử lý dữ
liệu}
  begin                                           { yêu cầu mở khoá }
    Op  $\leftarrow$  pm.opn
    res  $\leftarrow$  pm.result
    T  $\leftarrow$  pm.tid
    tìm đơn vị khóa lu sao cho  $x \subseteq lu$ , giải phóng khóa trên lu do T giữ
    if không còn khóa nào trên lu and
      có những thao tác đang đợi khóa lu trong hàng đợi then
      begin
        SOP  $\leftarrow$  thao tác đầu tiên trong hàng đợi
        SOP  $\leftarrow$  SOP  $\cup$  { O | O là một thao tác trên hàng đợi có thể khóa lu ở thể
thức khóa tương thích với các thao tác hiện hành trong SOP }
        đặt các khóa trên lu cho các thao tác trong SOP
        for tất cả các phép toán trong SOP do
          gửi mỗi thao tác đến bộ xử lý dữ liệu
        end-for
      end-if
    end
  end-case
until forever
```

end. (Basic LM)

Không may là, thuật toán khóa được cho trong Thuật toán 6.1 không đồng bộ hóa chính xác các thực thi giao dịch. Điều này là do khi tạo ra các lịch khả tuần tự, các thao tác khóa và giải phóng khóa cũng cần phải được điều phối. Chúng ta minh họa nó bằng thí dụ sau.

Thí dụ 7

Xét hai giao tác sau đây: $x = 50, y = 20$ 

$T_1, T_2 \implies x = 102, y = 38$
 $T_2, T_1 \implies x = 101, y = 39$
 $S \implies x = 102, y = 39$

$T_1:$	Read(x)	$T_2:$	Read(x)
	$x \leftarrow x + 1$		$x \leftarrow x * 2$
	Read(y)		Read(y)
	$Y \leftarrow y - 1$		$Y \leftarrow y * 2$
	Write(y)		Write(y)
	Commit		Commit

Dưới đây là một lịch hợp lệ được bộ quản lý khóa tạo ra khi sử dụng Thuật toán 6.1:

$$S = \{wl_1(x), R_1(x), W_1(x), lr_1(x), wl_2(x), R_2(x), W_2(x), lr_2(x), wl_2(y), R_2(y), W_2(y), lr_2(y), C_2, wl_1(y), R_1(y), W_1(y), lr_1(y), C_1\}$$

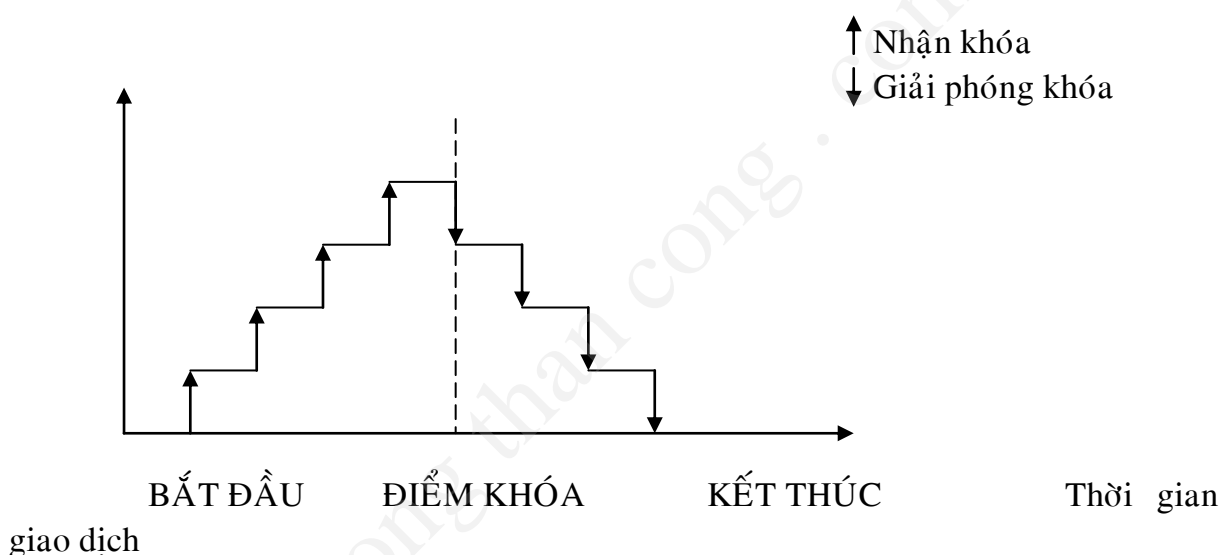
Ở đây, $lr_i(z)$ biểu thị thao tác giải phóng khóa trên z đang được T_i giữ.

Chú ý rằng S không khả tuần tự. Chẳng hạn nếu trước lúc thực hiện các giao tác này, giá trị của x và y lần lượt là 50 và 20, chúng ta hy vọng rằng giá trị sau khi thực hiện tương ứng là 102 và 38 nếu T_1 thực hiện trước T_2 , hoặc là 101 và 39 nếu T_2 thực hiện trước T_1 . Tuy nhiên kết quả thực hiện S cho ra giá trị của x và y lần lượt là 102 và 39. Rõ ràng S không khả tuần tự.

Vấn đề của lịch S trong thí dụ 7 là, thuật toán khóa chốt đã giải phóng các khóa được một giao tác giữ (chẳng hạn T_1) ngay khi lệnh đi kèm (đọc hoặc ghi) được thực hiện, và đơn vị khóa (chẳng hạn x) không cần truy xuất nữa. Tuy nhiên bản thân giao tác đó đang khóa những mục khác (chẳng hạn Y) sau khi nó giải phóng khóa trên x . Dù rằng điều này dường như có lợi vì làm tăng khả năng hoạt động đồng thời, nó cho phép các giao tác đan xen với nhau, làm mất đi tính biệt lập và tính nguyên tử tổng thể. Đây chính là lập luận của *phương pháp khóa chốt hai pha* (two-phase locking, 2PL)

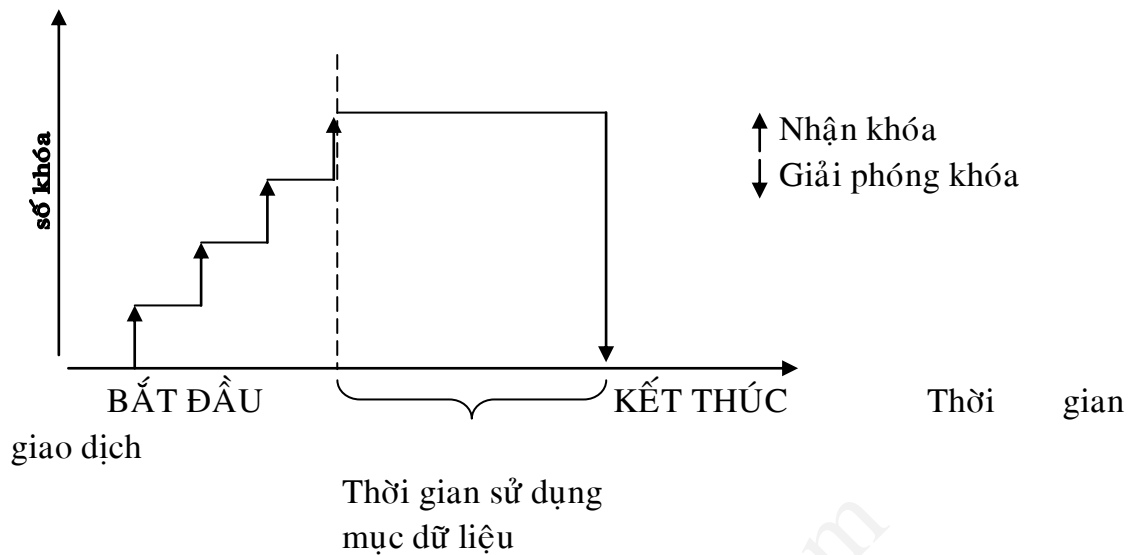
Chương 6: Điều khiển đồng thời phân tán

Qui tắc khóa hai pha chỉ đơn giản khẳng định rằng không có giao tác nào yêu cầu khóa sau khi nó đã giải phóng một trong các khóa của nó. Điều đó nói rằng một giao tác không được giải phóng khóa cho đến khi nó bảo đảm rằng không yêu cầu thêm khóa nữa. Các thuật toán 2PL thực hiện các giao tác qua hai pha. Mỗi giao tác có một *pha tăng trưởng* (growing phase), trong pha này nó nhận các khóa truy xuất các mục dữ liệu, và có một *pha thu hồi* (shrinking phase), là giai đoạn nó giải phóng các khóa (Hình 6.7). *Điểm khóa* (lockpoint) là thời điểm giao tác đã nhận được tất cả các khóa nhưng chưa bắt đầu giải phóng bất kỳ khóa nào. Vì thế điểm khóa xác định cuối pha tăng trưởng và khởi điểm pha thu hồi của một giao dịch. Một định lý nổi tiếng [Eswaran et al., 1976] khẳng định rằng mọi lịch tạo bởi một thuật toán điều khiển đồng thời tuân theo qui tắc 2PL đều khả tuần tự.



Hình 6.7. Biểu đồ khóa 2PL

Hình 6.7 chỉ ra rằng bộ quản lý khóa giải phóng các khóa ngay sau khi hoàn tất việc truy xuất. Điều này cho phép các giao tác đang đợi khóa tiếp tục tiến hành và nhận khóa, do vậy làm tăng hoạt động đồng thời. Tuy nhiên việc cài đặt gặp nhiều khó khăn bởi vì bộ quản lý khóa phải biết rằng giao tác đã nhận đủ tất cả mọi khóa và sẽ không cần khóa một mục nào nữa. Bộ quản lý khóa cũng phải biết rằng giao tác không còn cần truy xuất nữa mục dữ liệu đó nữa, vì thế khóa có thể được giải phóng. Cuối cùng nếu giao tác bị hủy bỏ sau khi giải phóng một khóa, nó có thể làm hủy bỏ luôn cả giao tác đã truy xuất các mục đã mở khóa. Hiện tượng này được gọi là *hủy bỏ dây chuyền* (cascading abort). Vì những khó khăn đó, phần lớn các bộ xếp lịch 2PL đều cài đặt một dạng khắt khe hơn có tên là *khóa chốt hai pha nghiêm ngặt* (strict two-phase locking) trong đó nó giải phóng toàn bộ các khóa vào lúc giao tác kết thúc (ủy thác hoặc hủy bỏ). Biểu đồ khóa loại này được trình bày trong Hình 6.8



Hình 6.8. Biểu đồ khóa hai pha nghiêm ngặt.

Bộ quản lý khóa 2PL nghiêm ngặt chỉ sửa lại một ít trong thuật toán 6.1. Thực sự chỉ cần sửa đổi phần xử lý các hồi đáp từ bộ xử lý dữ liệu nhằm bảo đảm rằng các khóa chỉ được giải phóng nếu thao tác là ủy thác hoặc hủy bỏ. Để cho đầy đủ, chúng tôi trình bày toàn bộ thuật toán 2PL nghiêm ngặt trong thuật toán 6.2. Thuật toán quản lý giao tác để xếp lịch theo 2PL được cho trong Thuật toán 6.3.

Thuật toán 6.2 S2PL-LM

declare-var

msg: Message
dop: Dbop
Op: Operation
x: DataItem
T: TransactionId
pm: Dpmsg
res: DataVal
SOP: OpSet

begin

repeat

WAIT(msg)

Case of msg

Dbop:

Begin

Op \leftarrow dop.opn

x \leftarrow dop. Data

T \leftarrow dop.tid

Chương 6: Điều khiển đồng thời phân tán

case of Op

Begin_transaction, Abort **or** Commit:

begin

gửi dop cho bộ xử lý dữ liệu

end

Read **or** Write

begin

tìm đơn vị khóa lu sao cho $x \subseteq lu$

if lu chưa khóa **or** thể thức khóa của lu tương thích với Op **then**

begin

đặt khóa trên lu ở thể thức thích hợp

gửi dop đến bộ xử lý dữ liệu

end

else

đặt dop vào một hàng đợi cho lu

end-if

end

end-case

Dpmsg:

begin

Op \leftarrow pm.opn

res \leftarrow pm.result

T \leftarrow pm.tid

if Op = Abort **or** Op = Commit **then**

begin

for mỗi đơn vị khóa lu bị khóa bởi T **do**

begin

giải phóng khóa trên lu do T giữ

if không còn khóa nào trên lu **and**

có các thao tác đang đợi trong hàng đợi cho lu **then**

begin

SOP \leftarrow thao tác đầu tiên trên hàng đợi

SOP \leftarrow SOP \cup {O | O là một thao tác trên hàng đợi có thể khóa lu
ở một thể thức tương thích với thao tác hiện tại trong SOP}

đặt các khóa trên lu cho các thao tác trong SOP

for tất cả các thao tác trong SOP **do**

gửi mỗi thao tác đến bộ xử lý dữ liệu

end-for

end-if

end-for

end-if

end

end-case

until forever

end. {S2PL-LM}

Thuật toán 6.3 Bộ quản lý giao tác 2PL (2PL-TM)

Declare-var

msg: Messenger

Op: Operation

x: DataItem

T: TransactionId

O: Dbop

sm: Scmsg

res: DataVal

SOP: OpSet

begin

repeat

WAIT(msg)

case of msg

Dbop:

begin

gửi O đến LM

end

Scmsg:

begin

Op \leftarrow sm.opn

res \leftarrow sm.result

T \leftarrow sm.tid

case of Op

Read:

begin

trả res về cho ứng dụng của người sử dụng (nghĩa là giao dịch)

end

Write:

begin

thông tin cho ứng dụng về việc hoàn tất hành động ghi

trả res về cho ứng dụng

end

Commit:

begin

hủy vùng làm việc của T

thông tin cho ứng dụng biết về việc hoàn tất thành công các giao tác

T

end

Abort:

begin

thông tin cho ứng dụng biết về việc hoàn tất hủy bỏ giao tác T

end

end-case

end

end-case

until forever

end. {2PL-TM}

Cần chú ý rằng mặc dù thuật toán 2PL cưỡng chế tính khả tuần tự tương tranh, nó không cho phép tất cả mọi lịch có tính khả tuần tự tương tranh. Xét thử lịch sau đây:

$$S = w_1(x)r_2(x)r_3(y)w_1(y)$$

S không được thuật toán 2PL cho phép dùng vì T_1 cần thu một khóa ghi trên y sau khi nó giải phóng khóa ghi của nó trên x. Tuy nhiên đây là lịch khả tuần tự theo thứ tự $T_3 \leftarrow T_1 \leftarrow T_2$. Thứ tự khóa có thể được tận dụng để thiết kế các thuật toán khóa cho phép những lịch thuộc loại này.

Ý tưởng chính nằm ở chỗ trước tiên cần nhận xét rằng trong lý thuyết khả tuần tự, thứ tự tuần tự hóa các thao tác tương tranh cũng quan trọng như việc phát hiện tương tranh và điều này có thể được tận dụng khi định nghĩa các thể thức khóa. Hệ quả là ngoài các khóa đọc (dùng chung, shared) và khóa ghi (độc quyền, exclusive), chúng ta có thể định nghĩa một thể thức khóa thứ ba: *dùng chung có thứ tự* (ordered shared). Khóa dùng chung có thứ tự của một đối tượng x bởi các giao tác T_i và T_j mang ý nghĩa như sau: cho một lịch S có các khóa dùng chung có thứ tự giữa các thao tác $o \in T_i$ và $p \in T_j$, nếu T_i thu được khóa o trước khi T_j thu được khóa p thì o được thực thi trước p. Xét bảng tương thích giữa các khóa đọc và ghi đã cho trong hình 6.5. Nếu có thêm khóa dùng chung có thứ tự thì có tám biến thể của bảng này. Hình 6.5 chỉ là một trong số đó và hình 6.9 trình bày thêm hai biến thể nữa. Thí dụ trong hình 6.9(a) có một mối liên hệ dùng chung có thứ tự giữa $rl_j(x)$ và $w_l(x)$ [ký hiệu là $rl_j(x) \Rightarrow w_l(x)$] chỉ ra rằng T_i có thể thu được một khóa ghi trên x trong khi T_j giữ một khóa đọc trên x với điều kiện có một mối liên hệ dùng chung có thứ tự từ $rl_j(x)$ đến $w_l(x)$. Tám bảng tương thích có thể được so sánh với nhau ứng với tính chất được phép của chúng (nghĩa là ứng với các lịch sinh ra nhờ chúng), tạo ra một dàn các bảng sao cho bảng trong hình 6.5 là hạn chế nhất và bảng trong Hình 6.9(b) là tự do nhất.

	$rl_i(x)$	$wl_i(x)$		$rl_i(x)$	$wl_i(x)$
$rl_j(x)$	tương thích dùng chung	không tương thích		$rl_j(x)$	tương thích
$wl_j(x)$	dùng chung dùng chung có thứ tự	không tương thích		$wl_j(x)$	có thứ tự dùng chung
				có thứ tự	có thứ tự
		(a)			(b)

Hình 6.9. Bảng tương thích có thể thức khóa dùng chung có thứ tự.

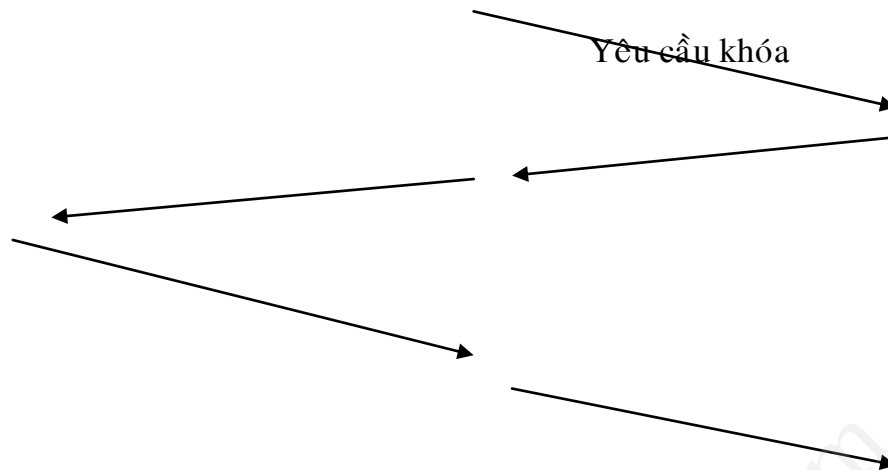
Trong thí dụ ở trên, khóa ghi dành cho T_i được gọi là đang được giữ (on hold) vì nó thu được sau khi T_j đã thu được khóa đọc trên x . Nghi thức khóa cưỡng chế một ma trận tương thích có chứa các thể thức khóa dùng chung có thứ tự hoàn toàn giống với 2PL, ngoại trừ là giao tác không giải phóng bất kỳ khóa nào khi một trong các khóa của chúng còn đang được giữ. Bằng không sẽ xảy ra các thứ tự tuần tự hóa lẫn lộn.

6.3.1 Nghi thức 2PL tập trung

Thuật toán 2PL được thảo luận trong phần trước có thể dễ dàng được mở rộng cho môi trường phân tán (nhân bản hoặc phân hoạch). Một cách để làm điều này là trao trách nhiệm quản lý khóa cho một vị trí duy nhất, nghĩa là chỉ có một vị trí là có bộ quản lý khóa. Các bộ quản lý giao tác ở các vị trí khác phải giao tiếp với nó chứ không phải với bộ quản lý khóa riêng của chúng. Cách tiếp cận này được gọi là thuật toán 2PL vị trí chính (primary site).

Truyền giao giữa các vị trí hiệp tác khi thực hiện một giao tác tuân theo thuật toán 2PL tập trung (centralized 2PL hay C2PL) được trình bày trong hình 6.10. Truyền giao này xảy ra giữa bộ quản lý giao tác tại vị trí khởi đầu giao tác (được gọi là **Transaction Manager** điều phối), bộ quản lý khóa tại vị trí trung tâm, và các bộ xử lý dữ liệu tại các vị trí có tham gia. Các vị trí tham gia là những vị trí có những thao tác xảy ra. Thứ tự các thông báo cũng được trình bày trong hình đó.

Một khác biệt quan trọng giữa thuật toán TM tập trung và thuật toán TM của Thuật toán 6.3 đó là TM phân tán phải cài đặt nghi thức điều khiển bản sao nếu CSDL được nhân bản. Thuật toán C2PL-LM cũng khác với bộ quản lý khóa 2PL nghiêm ngặt ở một điểm. Bộ quản lý khóa trung tâm không gửi các thao tác đến các bộ xử lý dữ liệu tương ứng; việc này do TM điều phối thực hiện.



Hình 6.10. Cấu trúc truyền giao của 2PL tập quyền.

Thuật toán quản lý giao tác 2PL tập trung (C2PL-TM) được trình bày trong Thuật toán 6.4 và có đưa thêm những thay đổi này vào, còn thuật toán quản lý khóa 2PL tập trung (C2PL-LM) được trình bày trong Thuật toán 6.5.

Thuật toán 6.4 Bộ quản lý giao tác 2PL tập trung (C2PL-TM)

Declare-var

T: TransactionId

Op: Operation

x: DataItem

msg: Message

O: Dbop

pm: Dpmsg

res: DataVal

S: SiteSet

begin

repeat

WAIT(msg)

case of msg

Dbop:

begin

Op \leftarrow O.opn

x \leftarrow O.data

T \leftarrow O.tid

case of Op

Begin_transaction:

Chương 6: Điều khiển đồng thời phân tán

```
begin
  S  $\leftarrow \emptyset$ 
end
Read:
begin
  S  $\leftarrow S \cup \{ \text{vị trí lưu x và chi phí truy xuất đến x là nhỏ nhất} \}$ 
  gửi O cho bộ quản lý khóa trung tâm
end
Write:
begin
  S  $\leftarrow S \cup \{ S_i \mid x \text{ được lưu tại } S_i \}$ 
  gửi O cho bộ quản lý khóa trung tâm
end
Abort or Commit
begin
  gửi O cho bộ quản lý khóa trung tâm
end
end-case
end
Scmsg:      {yêu cầu khóa được trao khi các khóa được giải phóng}
begin
  if yêu cầu khóa được trao then
    gửi O cho các bộ xử lý dữ liệu trong S
  else
    thông tin cho người dùng biết về việc kết thúc giao dịch
  end-if
end
Dpmsg:
begin
  Op  $\leftarrow$  pm.Opn
  res  $\leftarrow$  pm.result
  T  $\leftarrow$  pm.tid
case of Op
  Read:
    begin
      trả res về cho ứng dụng người sử dụng (nghĩa là giao dịch)
    end
  Write:
    begin
      thông tin cho ứng dụng biết về việc hoàn tất thao tác ghi.
    end
  Commit:
```

```
begin
  if tất cả các thành viên đều nhận được thông báo ủy thác then
    begin
      thông tin cho ứng dụng biết về việc hoàn tất thành công giao tác
      gửi pm cho bộ quản lý khóa trung tâm
    else {đợi cho đến khi tất cả đều nhận được thông báo ủy thác}
      ghi nhận sự kiện thông báo ủy thác đến các vị trí
    end-if
  end
Abort:
begin
  thông tin cho ứng dụng biết về việc hủy bỏ giao tác T
  gửi pm đến bộ quản lý khóa trung tâm
end
end-case
end
end-case
until forever
end. {C2PL-TM}
```

Thuật toán 6.5 Bộ quản lý khóa 2PL tập trung (C2PL-LM)

Declare-var

msg: Message
dop: SingleOp
Op: Operation
x: DataItem
T: TransactionId
SOP: OpSet

begin

repeat

WAIT(msg) {Thông báo chỉ có thể do TM điều phối gửi đến}

Op \leftarrow dop.opn

x \leftarrow dop.data

T \leftarrow dop.tid

case of Op

Read **or** Write:

begin

tìm đơn vị khóa lu cho $x \subseteq lu$

If lu chưa khóa **or** thể thức khóa của ly tương thích với Op **then**

begin

đặt khóa trên lu ở thể thức thích hợp

msg \leftarrow “Khóa được trao cho thao tác dop”

Chương 6: Điều khiển đồng thời phân tán

```
    gửi msg đến TM điều phối của T
end
else
    đặt Op vào một hàng đợi cho lu
end-if
end
Abort or Commit:
begin
    for mỗi đơn vị khóa lu bị khóa T do
begin
    giải phóng khóa trên lu do T giữ
    if còn những thao tác đang đợi lu trong hàng đợi then
begin
        SOP  $\leftarrow$  thao tác đầu tiên (gọi O) từ hàng đợi
        SOP  $\leftarrow$  SOP  $\cup$  {O | O là một thao tác trên hàng đợi có thể khóa lu ở
            thể thức tương thích với các thao tác trong SOP}
        đặt các khóa trên lu cho các thao tác trong SOP
        for tất cả các thao tác O trong SOP do
begin
            msg  $\leftarrow$  “Khóa được trao cho thao tác O”
            gửi msg đến tất cả các TM điều phối
        end-for
    end-if
end-for
msg  $\leftarrow$  “Các khóa của T đã giải phóng”
gửi msg đến TM điều phối của T
end
end-case
until forever
end. {C2PL-LM}
```

Một yếu điểm hay gặp của các thuật toán C2PL là có thể tạo ra một điểm ùn tắc quanh vị trí trung tâm. Hơn nữa hệ thống sẽ kém thích ứng (độ khả tín thấp) bởi vì sự cố hoặc tình trạng bất khả truy đến vị trí trung tâm có thể dẫn đến các sự cố hệ thống. Đã có những nghiên cứu chỉ ra rằng điểm ùn tắc thực sự sẽ hình thành khi tốc độ giao tác tăng lên, nhưng không đáng kể nếu tốc độ giao tác thấp. Thế nhưng người ta cũng thấy sự suy giảm hiệu năng khi tải trọng tăng cao trong các thuật toán dựa trên khóa chốt.

6.3.2 Thuật toán 2PL bản chính

Khóa chốt hai pha bản chính là sự mở rộng tầm thường của khóa chốt hai pha tập quyền với nỗ lực giải quyết các vấn đề về hiệu năng đã được thảo luận ở trên. Về

Chương 6: Điều khiển đồng thời phân tán

cơ bản, nó cài đặt các bộ quản lý khóa tại một số vị trí, trao trách nhiệm quản lý khóa trên một tập đơn vị khóa cho mỗi bộ quản lý. Sau đó bộ quản lý giao tác sẽ gửi các yêu cầu khóa và mở khóa đến các bộ quản lý khóa chịu trách nhiệm về đơn vị khóa đó. Thuật toán sẽ xử lý một bản của mỗi mục dữ liệu như bản chính của nó.

Chúng tôi không trình bày chi tiết thuật toán 2PL bản chính vì những thay đổi so với thuật toán 2PL tập quyền rất ít. Về cơ bản, thay đổi duy nhất là các nơi đặt bản chính phải được xác định cho mỗi mục trước khi gửi yêu cầu khóa hoặc mở khóa đến bộ quản lý khóa tại vị trí đó.

Thuật toán 2PL bản chính đã được đề xuất cho phiên bản phân tán thử nghiệm của hệ INGRES. Dù rằng nó đòi hỏi phải có một thư mục phức tạp tại mỗi vị trí, nó đã giảm được tải trọng cho vị trí trung tâm mà không phải trao đổi quá nhiều giữa các bộ quản lý giao tác và các bộ quản lý khóa. Về một nghĩa nào đó, đây là một bước trung gian giữa thuật toán 2PL tập quyền đã được thảo luận trong phần trước và thuật toán 2PL phân quyền sẽ được thảo luận trong phần kế tiếp.

6.3.3 Thuật toán 2PL phân quyền

2PL *phân quyền* (distributed 2PL hay D2PL) mong muốn có sẵn các bộ quản lý khóa tại mỗi vị trí. Nếu CSDL không nhân bản, thuật toán 2PL phân quyền sẽ suy biến thành thuật toán 2PL bản chính. Nếu CSDL có nhân bản, giao tác sẽ cài đặt nghi thức điều khiển bản sao ROWA.

Truyền giao giữa các vị trí để thực hiện một giao tác theo nghi thức 2PL phân quyền được trình bày trong hình 6.11. Chú ý rằng hình 6.11 không trình bày việc áp dụng qui tắc ROWA.

Thuật toán quản lý giao tác 2PL phân quyền tương tự như 2PL-TM nhưng có hai sửa đổi chính. Các thông báo gửi đến bộ quản lý khóa của vị trí trung tâm trong C2PL-TM sẽ được gửi đến bộ quản lý khóa của tất cả các vị trí tham gia trong D2PL-TM. Khác biệt thứ hai là các thao tác không do TM điều phối chuyển đến các bộ xử lý dữ liệu nhưng do các bộ quản lý khóa tham gia chuyển đi. Nghĩa là TM điều phối không chờ thông báo “yêu cầu khóa đã được trao”. Một điểm khác về hình 6.11 là, các bộ xử lý dữ liệu sẽ gửi thông báo “kết thúc thao tác” đến TM điều phối. Chọn lựa khác là mỗi bộ xử lý dữ liệu sẽ gửi thông báo đó cho bộ quản lý khóa của riêng nó rồi bộ quản lý khóa sẽ giải phóng khóa và thông tin cho TM điều phối. Chúng ta đã giải quyết định mô tả theo cách thứ nhất vì nó dùng một thuật toán quản lý khóa giống với bộ quản lý khóa 2PL nghiêm ngặt đã được thảo luận và nó làm cho việc thảo luận các nghi thức ủy thác đơn giản hơn (xem Chương

Chương 6: Điều khiển đồng thời phân tán

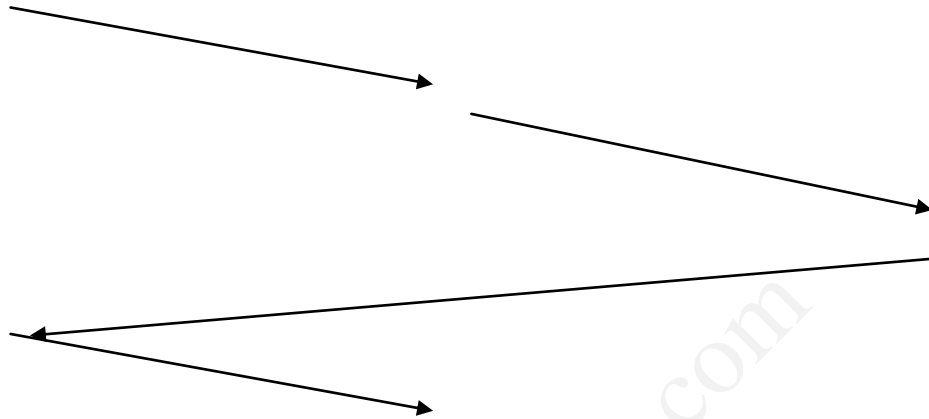
12). Do những tương đồng này, chúng ta không đưa ra các thuật toán TM và LM phân quyền ở đây. Các thuật toán 2PL phân quyền được dùng trong System R*.

Các bộ xếp lịch

TM điều phối

có tham gia

Các DM có tham gia



Hình 6 .11. Cấu trúc truyền giao của 2PL phân quyền.