**Nhân bản dữ liệu**

**6, Nhân bản dữ liệu**

Như chúng ta đã thảo luận trong các chương trước, cơ sở dữ liệu phân tán thường được nhân bản. Mục đích của việc nhân bản rất đa dạng:

**1.** Tính khả dụng của hệ thống. Như đã thảo luận trong Chương. 1, các DBMS phân tán có thể loại bỏ các điểm lỗi cục bộ bằng cách nhân bản dữ liệu, để các mục dữ liệu có thể truy cập được từ nhiều trang. Do đó, ngay cả khi một số trang web ngừng hoạt động, dữ liệu vẫn có thể truy cập được từ các trang web khác.

**2**. Hiệu suất. Như chúng ta đã thấy trước đây, một trong những nguyên nhân chính gây ra thời gian phản hồi là chi phí liên lạc. Nhân bản cho phép chúng ta định vị dữ liệu gần hơn với các điểm truy cập của chúng, do đó cục bộ hóa hầu hết quyền truy cập góp phần giảm thời gian phản hồi.

Localizing:cục bộ hóa (hạn chế quyền truy cập)

overhead la` những gi` ma` không muô'n chi phí, nhưng tránh không khỏi

**3**. Khả năng mở rộng. Khi các hệ thống phát triển về mặt địa lý(không gian) và về số lượng trang web (do đó, về số lượng yêu cầu truy cập), việc nhân bản cho phép hỗ trợ sự phát triển này với thời gian phản hồi có thể chấp nhận được.

**4**. Yêu cầu về ứng dụng. Cuối cùng, việc nhân bản có thể được quyết định bởi các ứng dụng, những ứng dụng này có thể muốn duy trì nhiều bản sao dữ liệu như một phần của thông số kỹ thuật hoạt động của chúng.

Mặc dù nhân bản dữ liệu có những lợi ích rõ ràng, nhưng nó đặt ra thách thức đáng kể trong việc giữ cho các bản sao khác nhau được đồng bộ hóa. Chúng ta sẽ sớm thảo luận vấn đề này, nhưng trước tiên chúng ta hãy xem xét mô hình thực hiện trong các cơ sở dữ liệu được nhân bản. Mỗi mục dữ liệu được nhân bản x có số lượng bản sao x1, x2,...,xn. Chúng tôi sẽ gọi x là mục dữ liệu logic và các bản sao (hoặc bản sao) 1 của nó là mục dữ liệu vật lý. Nếu tính minh bạch của bản sao được cung cấp, các giao dịch của người dùng sẽ thực hiện các thao tác đọc và ghi trên mục dữ liệu logic x. Giao thức kiểm soát bản sao chịu trách nhiệm ánh xạ các thao tác này để đọc và ghi trên các mục dữ liệu vật lý x1,...,xn. Như vậy, hệ thống hoạt động như thể có một bản sao duy nhất của mỗi mục dữ liệu—được gọi là hình ảnh hệ thống đơn lẻ hoặc tương đương một bản sao. Việc triển khai cụ thể các giao diện Đọc và Viết của trình giám sát giao dịch khác nhau tùy theo giao thức nhân bản cụ thể và chúng tôi sẽ thảo luận về những khác biệt này trong các phần thích hợp. Có một số quyết định và yếu tố ảnh hưởng đến việc thiết kế các giao thức nhân bản. Một số trong số này đã được thảo luận trong các chương trước, trong khi những điều khác sẽ được thảo luận ở đây.

• **Thiết kế cơ sở dữ liệu.** Như đã thảo luận trong Chương. 2, cơ sở dữ liệu phân tán có thể được nhân bản toàn bộ hoặc một phần. Trong trường hợp cơ sở dữ liệu được nhân bản một phần, số lượng mục dữ liệu vật lý cho mỗi mục dữ liệu logic có thể khác nhau và một số mục dữ liệu có thể thậm chí không được nhân bản. Trong trường hợp này, các giao dịch chỉ truy cập các mục dữ liệu không được nhân bản là các giao dịch cục bộ (vì chúng có thể được thực hiện cục bộ tại một địa điểm) và việc thực hiện chúng thường không liên quan đến chúng tôi ở đây. Các giao dịch truy cập các mục dữ liệu nhân bản phải được thực hiện tại nhiều địa điểm và chúng là giao dịch toàn cầu

• **Tính nhất quán của cơ sở dữ liệu**. Khi các giao dịch toàn cầu cập nhật các bản sao của một mục dữ liệu tại các trang web khác nhau, giá trị của các bản sao này có thể khác nhau tại một thời điểm nhất định . Một cơ sở dữ liệu nhân bản được cho là nhất quán lẫn nhau nếu tất cả các bản sao của từng mục dữ liệu của nó có giá trị giống hệt nhau. Điều khác biệt giữa các tiêu chí nhất quán lẫn nhau khác nhau là mức độ đồng bộ chặt chẽ của các bản sao. Một số đảm bảo rằng các bản sao nhất quán lẫn nhau khi một giao dịch cập nhật được thực hiện; do đó, chúng thường được gọi là tiêu chí nhất quán mạnh mẽ. Những cái khác có cách tiếp cận thoải mái hơn và được gọi là tiêu chí nhất quán yếu.

• **Nơi cập nhật được thực hiện**. Một quyết định thiết kế cơ bản trong thiết kế một giao thức nhân bản là nơi các bản cập nhật cơ sở dữ liệu được thực hiện lần đầu tiên. Các kỹ thuật có thể được mô tả là tập trung nếu chúng thực hiện cập nhật trước trên bản sao chính, so với phân tán nếu chúng cho phép cập nhật trên bất kỳ bản sao nào. Các kỹ thuật tập trung có thể được xác định thêm là bản chính đơn khi chỉ có một bản sao cơ sở dữ liệu chính trong hệ thống hoặc bản sao chính khi bản sao chính của từng mục dữ liệu có thể khác nhau.

• **Lan truyền cập nhật**. Sau khi các bản cập nhật được thực hiện trên một bản sao (chính hoặc bản sao khác), quyết định tiếp theo là cách các bản cập nhật được lan truyền đến những bản sao khác. Các lựa chọn thay thế được xác định là eager so với lazy. Kỹ thuật eager thực hiện tất cả của các bản cập nhật trong ngữ cảnh của giao dịch toàn cầu đã bắt đầu hoạt động ghi. Do đó, khi giao dịch được xác nhận, các cập nhật của nó sẽ được áp dụng cho tất cả các bản sao. Mặt khác, các kỹ thuật lazy sẽ lan truyền các bản cập nhật sau khi giao dịch bắt đầu được thực hiện. Các kỹ thuật háo hức được xác định thêm tùy theo thời điểm chúng đẩy từng thao tác ghi sang các bản sao khác—một số đẩy từng thao tác ghi riêng lẻ, một số khác gộp các thao tác ghi và truyền chúng tại điểm commit.

Eager : háo hức, nhanh

Lazy : lười biếng, chậm

• **Mức độ minh bạch của bản sao**. Một số giao thức nhân bản yêu cầu mỗi ứng dụng người dùng phải biết trang web chính nơi các hoạt động giao dịch sẽ được gửi. Các giao thức này chỉ cung cấp tính minh bạch nhân bản hạn chế cho các ứng dụng của người dùng. Các giao thức khác cung cấp tính minh bạch nhân bản đầy đủ bằng cách liên quan đến TM tại mỗi trang web. Trong trường hợp này, các ứng dụng người dùng gửi các giao dịch đến các TM cục bộ của họ thay vì trang chủ.

Chúng tôi thảo luận về các vấn đề nhất quán trong cơ sở dữ liệu nhân bản trong Phần. 6.1 và phân tích ứng dụng cập nhật tập trung so với phân tán cũng như cập nhật các lựa chọn thay thế lan truyền trong Phần. 6.2. Điều này sẽ dẫn chúng ta đến một cuộc thảo luận về các giao thức cụ thể phần6.3. phần 6.4, chúng tôi thảo luận về việc sử dụng các nguyên mẫu giao tiếp nhóm trong việc giảm chi phí nhắn tin của các giao thức nhân bản. Trong các phần này, chúng tôi sẽ giả định rằng không có lỗi nào xảy ra để chúng tôi có thể tập trung vào các giao thức nhân bản. Sau đó, chúng tôi sẽ giới thiệu các lỗi và điều tra cách các giao thức được sửa đổi để xử lý các lỗi trong Phần. 6,5

**6.1 Tính nhất quán của cơ sở dữ liệu được nhân bản**

Có hai vấn đề liên quan đến tính nhất quán của cơ sở dữ liệu được nhân bản. Một là tính nhất quán lẫn nhau, như đã thảo luận ở trên, liên quan đến sự hội tụ của các giá trị của các mục dữ liệu vật lý tương ứng với một mục dữ liệu logic. Thứ hai là tính nhất quán của giao dịch như chúng ta đã thảo luận trong Chương. 5. Khả năng tuần tự hóa, mà chúng tôi đã giới thiệu là tiêu chí về tính nhất quán của giao dịch cần được viết lại trong trường hợp cơ sở dữ liệu được nhân bản. Ngoài ra, có những mối quan hệ giữa tính nhất quán lẫn nhau và tính nhất quán trong giao dịch. Trong phần này, trước tiên chúng ta thảo luận về các phương pháp nhất quán lẫn nhau và sau đó tập trung vào việc xác định lại tính nhất quán của giao dịch và mối quan hệ của nó với sự nhất quán lẫn nhau.

**6.1.1 Nhất quán lẫn nhau**

Như đã chỉ ra trước đó, tiêu chí nhất quán lẫn nhau cho cơ sở dữ liệu được nhân bản có thể mạnh hoặc yếu. Mỗi loại phù hợp với các lớp ứng dụng khác nhau với các yêu cầu về tính nhất quán khác nhau.

Tiêu chí nhất quán lẫn nhau mạnh mẽ yêu cầu tất cả các bản sao của mục dữ liệu phải có cùng giá trị khi kết thúc quá trình thực hiện giao dịch cập nhật. Điều này đạt được bằng nhiều cách khác nhau, nhưng việc thực hiện 2PC tại điểm cam kết của giao dịch cập nhật là một cách phổ biến để đạt được sự nhất quán mạnh mẽ lẫn nhau. Tiêu chí nhất quán lẫn nhau yếu không yêu cầu giá trị của các bản sao của một mục dữ liệu giống hệt nhau khi một giao dịch cập nhật kết thúc.

Điều cần thiết (quan trọng) là,nếu hoạt động cập nhật ngừng trong một thời gian, các giá trị cuối cùng sẽ trở nên giống hệt nhau. Điều này thường được gọi là tính nhất quán cuối cùng, đề cập đến thực tế là các giá trị bản sao có thể khác nhau theo thời gian, nhưng cuối cùng sẽ hội tụ. Thật khó để định nghĩa khái niệm này một cách chính thức hoặc chính xác, mặc dù định nghĩa sau đây của Saito và Shapiro có lẽ chính xác như người ta mong đợi: Một [mục dữ liệu] được nhân bản cuối cùng sẽ nhất quán khi nó đáp ứng các điều kiện sau, giả sử rằng tất cả các bản sao bắt đầu từ cùng một trạng thái ban đầu.

• Tại bất kỳ thời điểm nào, đối với mỗi bản sao, có một tiền tố [lịch sử] tương đương với tiền tố [lịch sử] của mọi bản sao khác. Chúng tôi gọi đây là tiền tố cam kết cho bản sao.

• Tiền tố cam kết của mỗi bản sao phát triển đơn điệu theo thời gian.

• Tất cả các hoạt động không bị hủy bỏ trong tiền tố đã cam kết đáp ứng các điều kiện tiên quyết của chúng.

• Đối với mọi hoạt động được gửi α, cuối cùng thì α hoặc [sự hủy bỏ của nó] sẽ được đưa vào tiền tố đã cam kết.

Cần lưu ý rằng định nghĩa về tính nhất quán cuối cùng này khá mạnh—đặc biệt là các yêu cầu về tiền tố lịch sử giống nhau tại bất kỳ thời điểm nào và tiền tố cam kết phát triển đơn điệu. Nhiều hệ thống tuyên bố cung cấp tính nhất quán cuối cùng sẽ vi phạm các yêu cầu này.

Khả năng tuần tự hóa Epsilon (ESR) cho phép một truy vấn xem dữ liệu không nhất quán trong khi các bản sao đang được cập nhật, nhưng yêu cầu các bản sao hội tụ thành một bản sao trạng thái tương đương sau khi các bản cập nhật được truyền tới tất cả các bản sao. Nó giới hạn lỗi trên các giá trị đã đọc bằng một giá trị epsilon ( ), được xác định theo số lần cập nhật (thao tác ghi) mà một truy vấn “bỏ lỡ”. Đưa ra một giao dịch (truy vấn) TQ chỉ đọc, đặt TU là tập hợp tất cả các giao dịch cập nhật đang thực hiện đồng thời với TQ. Nếu RS(TQ) WS(TU ) = ∅ (TQ đang đọc một số bản sao của một số mục dữ liệu trong khi một giao dịch trong TU đang cập nhật (có thể là một bản sao khác) của những mục dữ liệu đó), thì sẽ xảy ra xung đột đọc-ghi và TQ có thể đang đọc dữ liệu không nhất quán. Sự không nhất quán bị giới hạn bởi những thay đổi được thực hiện bởi TU . Rõ ràng, ESR không hy sinh tính nhất quán của cơ sở dữ liệu mà chỉ cho phép chỉ đọcgiao dịch (truy vấn) để đọc dữ liệu không nhất quán. Vì lý do này, người ta đã khẳng định rằng ESR không làm suy yếu tính nhất quán của cơ sở dữ liệu mà “kéo dài” nó.

Các giới hạn lỏng lẻo khác cũng đã được thảo luận. Người ta thậm chí còn đề xuất rằng người dùng nên được phép chỉ định các ràng buộc về độ mới phù hợp với các ứng dụng cụ thể và các giao thức nhân bản nên thực thi các ràng buộc này. Các loại giới hạn độ mới có thể được chỉ định như sau:

• **Ràng buộc về thời gian**. Người dùng có thể chấp nhận sự khác biệt của các giá trị bản sao vật lý trong một khoảng thời gian nhất định: xi có thể phản ánh giá trị của bản cập nhật tại thời điểm t, trong khi xj có thể phản ánh giá trị tại t − và điều này có thể được chấp nhận.

• **Các ràng buộc về giá trị**. Có thể chấp nhận giá trị của tất cả các mục dữ liệu vật lý trong một phạm vi nhất định của nhau. Người dùng có thể coi cơ sở dữ liệu là nhất quán lẫn nhau nếu các giá trị không phân kỳ nhiều hơn một lượng (hoặc tỷ lệ phần trăm) nhất định.

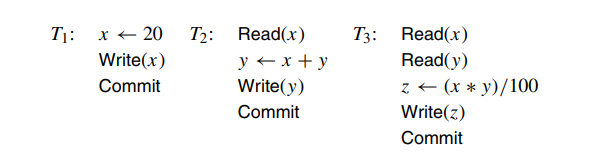
• **Ràng buộc trôi trên nhiều mục dữ liệu**. Đối với các giao dịch đọc nhiều mục dữ liệu, người dùng có thể hài lòng nếu thời gian trôi giữa các dấu thời gian cập nhật của hai mục dữ liệu nhỏ hơn một ngưỡng (nghĩa là chúng được cập nhật trong khoảng thời gian đó).ngưỡng) hoặc, trong trường hợp tính toán tổng hợp, nếu tổng hợp được tính toán trên một mục dữ liệu nằm trong một phạm vi nhất định của giá trị gần đây nhất (tức là, ngay cả khi các giá trị bản sao vật lý riêng lẻ có thể không đồng bộ hơn phạm vi này, như miễn là một tính toán tổng hợp cụ thể nằm trong phạm vi, nó có thể được chấp nhận).

Một tiêu chí quan trọng trong việc phân tích các giao thức sử dụng tiêu chí cho phép các bản sao phân kỳ là mức độ mới. Mức độ mới của một bản sao nhất định xi tại thời điểm t được định nghĩa là tỷ lệ các bản cập nhật đã được áp dụng tại xi tại thời điểm t trên tổng số bản cập nhật

**6.1.2 Tính nhất quán lẫn nhau so với tính nhất quán giao dịch**

Tính nhất quán lẫn nhau, như chúng tôi đã định nghĩa ở đây và tính nhất quán giao dịch như chúng tôi đã thảo luận trong Chương. 5 có liên quan, nhưng khác nhau. Tính nhất quán lẫn nhau đề cập đến các bản sao hội tụ về cùng một giá trị, trong khi tính nhất quán của giao dịch yêu cầu lịch sử thực thi toàn cầu phải được tuần tự hóa. Một DBMS được nhân bản có thể đảm bảo rằng các mục dữ liệu nhất quán lẫn nhau khi một giao dịch được thực hiện, nhưng lịch sử thực thi có thể không tuần tự hóa được trên toàn cầu(tổng quát,toàn bộ). Điều này được thể hiện trong ví dụ sau.

Ví dụ 6.1 Xem xét ba trang web (A, B và C) và ba mục dữ liệu (x, y, z)được phân bố như sau: site A chứa x, site B chứa x, y, site C chứa x, y, z. Chúng tôi sẽ sử dụng số nhận dạng trang web làm chỉ số dưới trên các mục dữ liệu để chỉ một bản sao cụ thể. Bây giờ hãy xem xét ba giao dịch sau:



Lưu ý rằng lệnh Write của T1 phải được thực hiện ở cả ba vị trí (vì x được nhân bản ở cả ba vị trí), lệnh Write của T2 phải được thực hiện tại B và C và lệnh Write của T3 chỉ được thực hiện tại C. Chúng ta đang giả sử một giao dịch mô hình thực thi trong đó các giao dịch có thể đọc các bản sao cục bộ của chúng nhưng phải cập nhật tất cả các bản sao. Giả sử rằng ba lịch sử địa phương(local:địa phương,cục bộ,thuộc về) sau đây được tạo ra tại các trang web:

Thứ tự tuần tự hóa trong HB là T1 → T2, trong khi ở HC là T2 → T3 → T1. Do đó, lịch sử toàn cục(local:toàn cục) không thể tuần tự hóa được. Tuy nhiên, cơ sở dữ liệu là nhất quán lẫn nhau. Ví dụ, giả sử rằng ban đầu xA = xB = xC = 10, yB = yC = 15 và zC = 7.

Với lịch sử trên, các giá trị cuối cùng sẽ là xA = xB = xC = 20, yB = yC = 35, zC = 3,5. Tất cả các bản sao vật lý (bản sao) thực sự đã hội tụ về cùng một giá trị

Tất nhiên, cơ sở dữ liệu có thể không nhất quán lẫn nhau và lịch sử thực thi không được tuần tự hóa trên toàn cầu, như được minh họa trong ví dụ sau.

Ví dụ 6.2 Xem xét hai trang web (A và B) và một mục dữ liệu (x) được nhân bản ở cả hai trang web (xA và xB). Xem xét thêm hai giao dịch sau:

Giả sử rằng hai lịch sử cục bộ sau được tạo tại hai địa điểm (một lần nữa sử dụng mô hình thực thi của ví dụ trước):

Mặc dù cả hai lịch sử này đều nối tiếp, nhưng chúng đánh số thứ tự T1 và T2 theo thứ tự ngược lại; do đó, lịch sử toàn cầu không thể tuần tự hóa được. Hơn nữa, xu hướng nhất quán lẫn nhau cũng bị vi phạm. Giả sử rằng giá trị của x trước khi thực hiện các giao dịch này là 1. Khi kết thúc thực hiện các lịch trình này, giá trị của x là 60 tại trang A, trong khi đó là 15 tại trang B. Do đó, trong ví dụ này, lịch sử toàn cầu là không thể tuần tự hóa và cơ sở dữ liệu không nhất quán lẫn nhau.

Với quan sát trên, tiêu chí nhất quán giao dịch được đưa ra trong Chương. 5 được mở rộng trong các cơ sở dữ liệu được nhân bản để xác định khả năng tuần tự hóa một bản sao. Khả năng tuần tự hóa một bản sao (1SR) cho biết tác động của các giao dịch đối với dữ liệu được nhân bản các mục phải giống như thể chúng đã được thực hiện lần lượt trên một tập hợp các mục dữ liệu. Nói cách khác, các lịch sử tương đương với một số thực hiện nối tiếp đối với các mục dữ liệu không được nhân bản.

Cách ly ảnh chụp nhanh mà chúng tôi đã giới thiệu trong Chương. 5 cũng đã được mở rộng cho cơ sở dữ liệu được nhân bản và được sử dụng như một tiêu chí nhất quán giao dịch thay thế trong ngữ cảnh của cơ sở dữ liệu được nhân bản. Tương tự, một dạng khả năng tuần tự hóa yếu hơn, được gọi là khả năng tuần tự hóa đồng thời thoải mái (RC-) đã được xác định tương ứng với mức cô lập đã cam kết đọc.

**6.2 Update Management Strategies(Cập nhật chiến lược quản lý)**

Như đã thảo luận trước đó, các giao thức nhân bản có thể được phân loại theo thời điểm các bản cập nhật được truyền tới các bản sao (eager so với lazy) và nơi cập nhật được phép xảy ra (centralized so với distributed). Hai quyết định này thường được gọi là chiến lược quản lý cập nhật. Trong phần này, chúng ta thảo luận về các lựa chọn thay thế này trước khi chúng ta trình bày các giao thức trong phần tiếp theo.

Note:

Eager : háo hức, hăng hái hoặc là dễ hiểu nhất thì có thể dịch là nhanh

Lazy : lười biếng hoặc là dễ hiểu nhất thì có thể dịch là chậm

Centralized : tập trung

Distributed : phân tán

**6.2.1 Eager Update Propagation(Lan truyền cập nhật Eager)**

Các phương pháp lan truyền bản cập nhật Eager áp dụng các thay đổi cho tất cả các bản sao trong bối cảnh của chuyển đổi cập nhật. Do đó, khi chuyển đổi cập nhật được thực hiện, tất cả các bản sao có cùng giá trị. Thông thường, các kỹ thuật lan truyền Eager sử dụng 2PC tại điểm commit, nhưng, như chúng ta sẽ thấy sau, các lựa chọn thay thế có thể đạt được thỏa thuận. Hơn nữa, lan truyền Eager có thể sử dụng lan truyền *synchronous* của mỗi bản cập nhật bằng cách áp dụng nó trên tất cả các bản sao cùng một lúc (khi Write được phát hành) hoặc lan truyền bị *deferred* theo đó các bản cập nhật được áp dụng cho một bản sao khi chúng được phát hành, nhưng ứng dụng của chúng trên các bản sao khác được chia theo đợt và hoãn lại cho đến khi kết thúc giao tác. Lan truyền bị trì hoãn có thể được triển khai bằng cách đưa các bản cập nhật vào thông báo *“Prepare-to-Commit”* khi bắt đầu thực hiện 2PC.

Các kỹ thuật Eager thường thực thi các tiêu chí nhất quán lẫn nhau mạnh mẽ. Vì tất cả các bản sao đều nhất quán lẫn nhau khi kết thúc một giao tác cập nhật, nên lần đọc tiếp theo có thể đọc từ bất kỳ bản sao nào (nghĩa là người ta có thể ánh xạ *R*(x) thành *R*(xi ) cho bất kỳ xi nào). Tuy nhiên, *W*(x) phải được áp dụng cho tất cả các bản sao (nghĩa là *W*(xi ), ∀xi ). Do đó, các giao thức tuân theo quá trình lan truyền cập nhật Eager được gọi là giao thức read-one/write-all (ROWA).

Ưu điểm của việc lan truyền cập nhật Eager có 3 điều. Đầu tiên, chúng thường đảm bảo rằng tính nhất quán lẫn nhau được thực thi bằng cách sử dụng 1SR; do đó, không có giao dịch không nhất quán. Thứ hai, một chuyển đổi có thể đọc một bản sao cục bộ của mục dữ liệu (nếu có sẵn một bản sao cục bộ) và chắc chắn rằng giá trị cập nhật được đọc. Vì vậy, không cần phải đọc từ xa. Cuối cùng, những thay đổi đối với bản sao được thực hiện một cách atomically; do đó, việc phục hồi sau sự cố có thể được điều chỉnh bởi các giao thức mà chúng ta đã nghiên cứu trong chương trước.

Nhược điểm chính của việc lan truyền cập nhật Eager là một giao tác phải cập nhật tất cả các bản sao trước khi nó có thể kết thúc. Điều này có hai hậu quả. Đầu tiên, hiệu suất thời gian phản hồi của chuyển đổi cập nhật bị ảnh hưởng, vì nó thường phải tham gia vào quá trình thực thi 2PC và do tốc độ cập nhật bị giới hạn bởi máy chậm nhất. Thứ hai, nếu một trong các bản sao không có sẵn, thì giao tác không thể chấm dứt vì tất cả các bản sao cần được cập nhật. Như đã thảo luận trong Chương. 5, nếu có thể phân biệt giữa lỗi trang web và lỗi mạng, thì người ta có thể chấm dứt giao tác miễn là chỉ có một bản sao không khả dụng (hãy nhớ rằng việc không có nhiều hơn một trang web khiến 2PC bị chặn), nhưng nói chung là không thể để phân biệt hai loại hư hỏng này.

*Note:*

*Synchronous : đồng bộ*

*Write : ghi*

*Deferred : trì hoãn*

*“Prepare-to-Commit” : “Chuẩn bị thực hiện”*

*read-one/write-all : đọc 1/ghi tất cả*

*atomically : nguyên tử (đặc tính nguyên tử trong database)*

**6.2.2 Lazy Update Propagation(Lan truyền cập nhập Lazy)**

Trong quá trình lan truyền bản cập nhật Lazy, các bản cập nhật bản sao không được thực hiện trong bối cảnh của giao tác cập nhật. Nói cách khác, giao tác không đợi cho đến khi các bản cập nhật của nó được áp dụng cho tất cả các bản sao trước khi thực hiện—giao tác này thực hiện ngay sau khi một bản sao được cập nhật. Việc truyền tới các bản sao khác được thực hiện *asynchronously* từ chuyển đổi ban đầu, bằng các *refresh transactions* được gửi đến các trang web sao chép một thời gian sau khi giao tác cập nhật được thực hiện. Một giao tác làm mới mang chuỗi cập nhật của giao tác cập nhật tương ứng.

Lan truyền Lazy được sử dụng trong những ứng dụng mà tính nhất quán mạnh mẽ lẫn nhau có thể không cần thiết và quá hạn chế. Các ứng dụng này có thể chấp nhận một số điểm không nhất quán giữa các bản sao để đổi lấy hiệu suất tốt hơn. Ví dụ về các ứng dụng như vậy là Domain Name Service (DNS), cơ sở dữ liệu trên các trang web được phân phối rộng rãi về mặt địa lý, cơ sở dữ liệu di động và cơ sở dữ liệu trợ lý kỹ thuật số cá nhân. Trong những trường hợp này, tính nhất quán lẫn nhau yếu được thi hành.

Ưu điểm chính của các kỹ thuật lan truyền cập nhật Lazy là chúng thường có thời gian phản hồi thấp hơn đối với các giao tác cập nhật, vì một giao tác cập nhật có thể thực hiện ngay sau khi nó cập nhật một bản sao. Nhược điểm là các bản sao không nhất quán lẫn nhau và một số bản sao có thể đã lỗi thời, do đó, thao tác đọc cục bộ có thể đọc dữ liệu cũ và không đảm bảo trả về giá trị cập nhật. Hơn nữa, trong một số tình huống mà chúng ta sẽ thảo luận sau, các giao tác có thể không nhìn thấy bản ghi của chính chúng, tức là *Ri*(x) của một giao tác cập nhật *Ti* có thể không nhìn thấy tác động của *Wi*(x) đã được thực hiện trước đó. Điều này đã được gọi là *transaction inversion*. Khả năng tuần tự hóa một bản sao mạnh (1SR mạnh) và snapshot độc lập mạnh (SI mạnh) ngăn chặn tất cả các đảo ngược giao tác ở các mức độc lập 1SR và SI tương ứng, nhưng việc cung cấp lại rất tốn kém. Các bảo đảm yếu hơn của 1SR và SI toàn cục, trong khi cung cấp ít tốn kém hơn nhiều so với các đối tác mạnh hơn của chúng, không ngăn được các đảo ngược giao tác. Bảo đảm giao tác ở cấp độ phiên ở cấp độ độc lập 1SR và SI đã được đề xuất để giải quyết những thiếu sót này bằng cách ngăn chặn đảo ngược giao tác trong phiên khách hàng nhưng không nhất thiết phải qua các phiên. Các bảo đảm cấp phiên này cung cấp ít tốn kém hơn so với các đối tác mạnh của chúng trong khi vẫn bảo tồn nhiều thuộc tính mong muốn của các đối tác mạnh.

Note:

*asynchronously* : không đồng bộ

*refresh transactions* : làm mới giao tác

*transaction inversion* : đảo ngược giao tác

**6.2.3 Centralized Techniques(Kỹ thuật tập trung)**

Các kỹ thuật lan truyền cập nhật tập trung yêu cầu các bản cập nhật trước tiên được áp dụng tại bản sao chính và sau đó được truyền tới các bản sao khác (được gọi là *slaves*). Trang web lưu trữ bản sao chính cũng được gọi là *master site*, trong khi các trang web lưu trữ bản sao phụ cho mục dữ liệu đó được gọi là *slave sites*.

Trong một số kỹ thuật, có một bản gốc duy nhất cho tất cả dữ liệu được sao chép. Chúng tôi gọi đây là các kỹ thuật tập trung *single master*. Trong các giao thức khác, bản sao chính cho mỗi mục dữ liệu có thể khác nhau (tức là, đối với mục dữ liệu x, bản sao chính có thể được xi lưu trữ tại trang Si, trong khi đối với mục dữ liệu y, nó có thể được lưu trữ yj tại trang Sj). Chúng thường được gọi là kỹ thuật tập trung *primary copy*.

Ưu điểm của các kỹ thuật tập trung là có 2 điều. Đầu tiên, việc áp dụng các bản cập nhật rất dễ dàng vì chúng chỉ xảy ra ở trang web chính và chúng không yêu cầu đồng bộ hóa giữa nhiều trang web bản sao. Thứ hai, có sự đảm bảo rằng ít nhất một trang web—trang web chứa bản sao chính—có các giá trị cập nhật cho một mục dữ liệu. Các giao thức này thường phù hợp trong kho dữ liệu và các ứng dụng khác nơi xử lý dữ liệu được tập trung tại một hoặc một số trang web chính.

Nhược điểm chính là, giống như trong bất kỳ thuật toán tập trung nào, nếu có một trang trung tâm lưu trữ tất cả các máy chủ, thì trang này có thể bị quá tải và có thể trở thành nút cổ chai. Phân phối trách nhiệm trang chủ cho mỗi mục dữ liệu như trong các kỹ thuật sao chép chính là một cách để giảm chi phí này, nhưng nó làm phát sinh các vấn đề về tính nhất quán, đặc biệt là đối với việc duy trì khả năng tuần tự hóa toàn cầu trong các kỹ thuật sao chép Lazy, do các giao tác làm mới phải được thực hiện tại các bản sao theo cùng một thứ tự tuần tự hóa. Chúng ta thảo luận về những điều này hơn nữa trong các phần có liên quan.

Note:

*Slaves* : bản sao phụ

*master site* : trang web chính

*slave sites* : trang web phụ

*single master* : chủ thể duy nhất

*primary copy* : sao chép chính

**6.2.4 Distributed Techniques(Kỹ thuật phân tán)**

Các kỹ thuật phân tán áp dụng bản cập nhật trên bản sao cục bộ tại nơi bắt đầu giao tác cập nhật và sau đó các bản cập nhật được truyền tới các trang web bản sao khác. Chúng được gọi là các kỹ thuật phân tán vì các giao tác khác nhau có thể cập nhật các bản sao khác nhau của cùng một hạng mục dữ liệu đặt tại các vị trí khác nhau. Chúng thích hợp cho các ứng dụng hợp tác với các trung tâm điều hành/ra quyết định phân tán. Chúng có thể phân tán tải đồng đều hơn và có thể cung cấp tính khả dụng cao nhất của hệ thống nếu được kết hợp với các kỹ thuật lan truyền Lazy.

Một sự phức tạp nghiêm trọng phát sinh trong các hệ thống này là các bản sao khác nhau của một mục dữ liệu có thể được cập nhật đồng thời tại các trang web khác nhau (master). Nếu các kỹ thuật phân tán được kết hợp bởi các phương pháp lan truyền Eager, thì các phương pháp kiểm soát phân tán đồng thời có thể giải quyết thỏa đáng vấn đề cập nhật đồng thời. Tuy nhiên, nếu sử dụng các phương pháp lan truyền Lazy, thì các giao tác có thể được thực hiện theo các thứ tự khác nhau tại các trang web khác nhau, gây ra lịch sử toàn cầu không phải 1SR. Hơn nữa, các bản sao khác nhau sẽ không đồng bộ. Để quản lý những vấn đề này, một phương pháp đối chiếu được áp dụng liên quan đến các giao tác hoàn tác và làm lại theo cách mà việc thực hiện giao tác giống nhau tại mỗi vị trí. Đây không phải là một vấn đề dễ dàng vì việc điều chỉnh thường phụ thuộc vào ứng dụng.

Note:

Master : bản chính

**6.3, Replication Protocol**

Trong phần trước, chúng ta đã thảo luận về hai hướng có thể phân loại các kỹ thuật quản lý cập nhật. Các hướng này là trực giao; do đó, có thể có bốn sự kết hợp.Chúng ta thảo luận về từng lựa chọn thay thế trong phần này. Để đơn giản hóa việc trình bày, chúng ta giả sử một cơ sở dữ liệu được sao chép đầy đủ, có nghĩa là tất cả các giao dịch cập nhật là toàn cục. Chúng tôi giả định thêm rằng mỗi trang web triển khai kỹ thuật kiểm soát tương tranh dựa trên 2PL.

**6.3.1 Giao thức Eager Centralized**

Site master điều khiển các thao tác trên các mục dữ liệu trong các giao thức Eager Centralized . Những giao thức này đi kèm với những kĩ thuật có tính nhất quán cao để các cập nhật 1 mục dữ liệu được áp dụng cho các bản sao của nó trong ngữ cảnh cập nhật giao dịch , sử dụng giao thức 2PC để commit các cập nhật (ngoài ra còn có thể dùng các giao thức khác). Sau khi cập nhật giao dịch hoàn tất, các bản sao sẽ nhất quán dữ liệu và lịch sử toàn cục sẽ đạt được 1SR.

2 thông số thiết kế được đề cập ở các phần trước sẽ quyết định việc cài đặt các giao thức Eeager centralized: Nơi thực hiện cập nhật, và độ minh bạch. Thông số đầu: lựa chọn giữa việc dùng 1 site master cho tất cả các mục dữ liệu (single master), hoặc dùng nhiều site master riêng biệt cho 1 hoặc 1 nhóm các mục dữ liệu (primary copy). Thông số thứ 2 để chỉ ra rằng liệu các ứng dụng có biết vị trí bản master (limited application transparency) hay phải phụ thuộc vào trình quản lý tiến trình cục bộ (local TM) để xác định vị trí bản master (full replication transparency).

**6.3.1.1 Single Master/Limited Replication Transparency**

Thiết kế đơn giản nhất là dùng 1 bản master cho toàn bộ csdl và giới hạn độ minh bạch( limited replication transparency). Trong trường hợp này, cập nhật giao dịch toàn cục (gồm ít nhất 1 thao tác W(x), x là dữ liệu dược nhân bản) được submit trực tiếp tới site master, cụ thể là tới trình quản lý tiến trình (TM) ở site master. Tại master, mỗi thao tác R(x) được thực hiện trên bản master (thao tác R(x) được chuyển thành R(xM), M kí hiệu là bản master) và được thực hiện như sau: 1 khóa đọc (read lock) được cấp cho dữ liệu xM, thực hiện thao tác đọc và trả về kết quả cho người dùng. Tương tự như vậy, mỗi thao tác W(x) sẽ cập nhật trên bản master khi nhận 1 khóa ghi (write lock) và thực hiện thao tác ghi. TM của master chuyển tiếp lệnh ghi tới các site slave (đồng bộ hoặc có trì hoãn) (Fig. 6.1).

Trong cả 2 trường hợp, điều quan trọng là phải thực hiện các cập nhật mang tính xung đột theo đúng thứ tự tại các slave như ở bản master. Ta có thể dùng timestamp hoặc các cách sắp xếp khác.

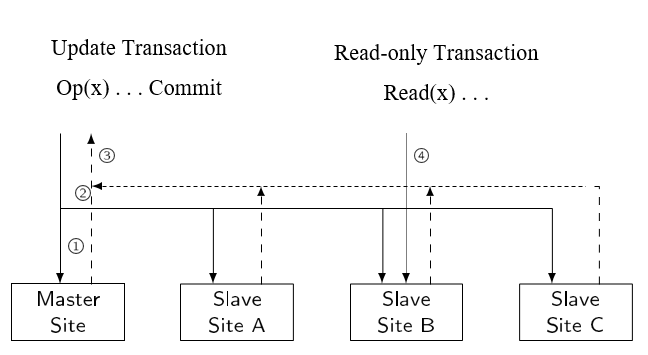


Fig. 6.1 (1) Lệnh Write được thực thi trên bản master;

(2) Write được chuyển tiếp tới các bản sao;

(3) Các cập nhật có hiệu lực tại thời điểm commit;

(4) Giao dịch chỉ đọc: gửi lệnh Read tới bất cứ bản sao slave cần thiết

Người dùng có thể submit giao dịch chỉ đọc tới bất cứ site slave nào. Giao dịch chỉ đọc tại các slave được thực hiện theo thuật toán kiểm soát truy cập đồng thời như C2PL. Thuật toán C2PL yêu cầu 1 số thay đổi nhỏ của TM tại các site không phải master để xử lý thao tác ghi. Khi 1 site slave nhận lệnh, nó chuyển tiếp đến site master để yêu cầu cấp khóa đọc. Thao tác đọc sẽ được thực hiện tại site master rồi trả kết quả cho ứng dụng, hoặc master có thể cấp khóa cho site yêu cầu khóa để thực hiện đọc cục bộ.

Ta có thể giảm bớt việc cho site master bằng cách đọc cục bộ mà kh cần nhận khóa đọc từ site master. Các thuật toán kiểm soát truy cập đồng thời cục bộ bảo đảm tranh chấp đọc-ghi cục bộ được tuần tự, và vì chỉ master mới có thể thực hiện thao tác ghi, tranh chấp ghi-ghi cục bộ sẽ không xảy ra. Tuy nhiên, thao tác đọc có thể đọc mục dữ liệu trước hoặc sau thi có cập nhật. 1 thao tác đọc ở site slave này có thể đọc giá trị trước khi được cập nhật, nhưng 1 site khác lại có thể đọc giá trị đã được cập nhật. Điều này là không đáng kể để đạt được lịch sử toàn cục 1SR. Ví dụ sau mô tả:

Example 6.3 1 mục dữ liệu x có site master ở site A và các bản sao ở site B và C. Có 3 thao tác được thực hiên:

| T1: | Write(x)  Commit | T2: | Read(x)  Commit | T3: | Read(x)  Commit |
| --- | --- | --- | --- | --- | --- |

Giả sử T2 được gửi tới slave tại site B và T3 tới site C. T2 đọc x tại B [R2(xB)] trước khi cập nhật tại T1 được chuyển tới B, còn T3 đọc x tại C

[R3(xC)] sau khi cập nhật tại T1 được chuyển tới C. Lịch sử 2 site slave là:

HB = {R2(x), C2, W1(x), C1}

HC = {W1(x), C1, R3(x), C3}

Thứ tự tại site B là T2 → T1, còn tại site là T1 → T3. Thứ tự toàn cục do đó là T2 → T1 → T3, vẫn thỏa mãn 1SR.

Nếu làm theo cách này, giao dịch đọc có thể đọc dữ liệu đanh được cập nhật tại master, nhưng vẫn đảm bảo lịch sử toàn cục là 1SR.

Trong giao thức thay thế này, khi 1 site slave Si nhận lệnh R(x), nó nhận 1 khóa đọc cục bộ, đọc cục bộ (R(xi)), và trả về kết quả. Khi nhận lệnh W(x), nếu là từ site master thì nó sẽ ghi trên bản sao cục bộ, còn nếu là từ người dùng thì sẽ từ chổi W(x).

Các giao thức thay thế của single master eager centralized dễ cài đặt nhưng 1 vấn đề cần thực hiện là làm sao để chỉ ra 1 giao dịch là cập nhât giao dịch hay là giao dịch chỉ đọc. Ta có thể làm được điều này bằng cách khai báo cụ thể trong lệnh Begin\_transaction.

**6.3.1.2 Single Master/Full Replication Transparency**

Giao thức Single master eager centralized yêu cầu mỗi người dùng phải biết vị trí site master và khiến cho khối lượng công việc master phải làm là khá lớn: thực hiện thao tác đọc các giao dịch, đóng vai trò điều phối viên cho các giao dịch. Ta có thể 1 phần xử lý vấn đề này bằng cách sử dụng TM cục bộ tại các site có giao dịch được thực hiện. Các giao dịch không được gửi thẳng tới site master, mà tới TM cục bộ. TM đóng vai trò điều phối cho các cập nhật giao dịch và giao dịch chỉ đọc. Các ứng dụng chuyển tiếp giao dịch tới TM cục bộ với độ minh bạch đầy đủ (full transparency).

Có những cách khác khai triển full transparency— TM điều phối chỉ đóng vai trò là 1 “router,” chuyển tiếp giao dịch trực tiếp tới master. Site master có thể thực hiện giao dịch cục bộ và trả về kết quả cho ứng dụng. Tuy cách khai triển này có độ minh bạch đầy đủ và có vài ưu điểm như dễ cài đặt, site master vẫn sẽ bị quá tải công việc. Ta có thể khai triển 1 cách khác như sau:

1. TM điều phối chuyển tiếp giao dịch tới master. Không đòi hỏi phải thay đổi thuật toán C2PL-TM (Algorithm 5.1).

2. Nếu thao tác là R(x), quản lý khóa tập trung (C2PL-LM Algorithm 5.2) đặt khóa đọc bản sao của x (bản gốc gọi là xM) và thông báo cho TM điều phối là đã cấp khóa đọc. TM điều phối chuyển tiếp R(x) tới bất kì site slave có bản sao của x [chuyển R(x) -> R(xi)]. Lệnh đọc được thực hiện bằng DP (data processor) của slave đó.

3. Nếu thao tác là W(x), quản lý khóa tập trung (master) sẽ thực hiện:

(a) Đầu tiên, đặt khóa ghi cho các bản sao xM.

(b) Sau đó. Nó gọi đến DP cục bộ để thực hiện W(xM) trên các bản sao.

(c) Cuối cùng, nó thông báo cho TM điều phối là đã cấp khóa ghi.

TM điều phối sẽ gửi W(x) tới mọi slaves có chứa bản sao của x; DP tại các slave thực hiện ghi cục bộ.

Điểm khác biệt là site master không cần xử lý lệnh đọc và không phải điều phối các cập nhật xảy ra trên bản sao. TM cục bộ sẽ làm điều đó.

Dễ dàng thấy được thuật toán này đảm bảo lịch sử toàn cục đạt được 1SR do thứ tự tuần tự được quyết định bởi 1 master. Thuật toán này cũng tuân thủ giao thức ROWA—do các bản sao được đảm bảo là đã cập nhật khi cập nhật giao dịch hoàn tất và lệnh đọc có thể được thực hiện ở bất kì bản sao nào.

Để mô tả cách thuật toán eager kết hợp kiểm soát bản sao (replica control) và kiểm soát đồng thời (concurrency control), ta có thuật toán Transaction Management

TM điều phối (Algorithm 6.1) và thuật toán Lock Management site master

(Algorithm 6.2)

Trong đoạn thuật toán dưới đây, LM chỉ đơn thuần gửi 1 message “Lock granted”, không phải là kết quả của cập nhật giao dịch. Khi cập nhật được chuyển tiếp tới các slave bới TM điều phối, các slave cần tự thực hiện thao tác cập nhật. Đây được gọi là chuyển đổi thao tác (operation transfer). 1 cách khai triển khác là thêm vào message “Lock granted” kết quả của cập nhật giao dịch, sau đó nó được chuyển tiếp tới slave cần được cập nhật. Đây được gọi là chuyển đổi cấu trúc (state *transfer)*. Tuy chỉ khác biệt nhỏ ở thao tác W(x), nhưng phải nhớ: thao tác ghi là trừu tượng; mỗi cập nhật giao dịch đòi hỏi việc thực thi các biểu thức SQL, trong đó việc phân biệt rõ ràng là vô cùng quan trọng.

Cách khai triển giao thức như trên giảm bớt gánh nặng cho site master, cắt giảm nhu cầu cần biết vị trí master của người dùng. Tuy nhiên, cách khai triển này phức tạp hơn cách khai triển thay thế đầu tiên được bàn phần trên. Cụ thể, TM cục bộ đóng vai trò điều phối viên cho 2PC và site master site trở thành 1 thành phần trong đó. Điều này đòi hỏi thay đổi các thuật toán tại các site.

Algorithm 6.1: Eager Single Master Modifications to C2PL-TM

begin

...

if lock request granted then

**if** op.Type = W then

S ← set of all sites that are slaves for the data item

else

S ← any one site which has a copy of data item

end if

DPS(op) {send operation to all sites in set S}

else

inform user about the termination of transaction

end if

...

end

**Algorithm 6.2:** Eager Single Master Modifications to C2PL-LM

begin

...

switch op.Type do

**case** R or W do {lock request; see if it can be granted}

find the lock unit lu such that op.arg ⊆ lu ;

if lu is unlocked or lock mode of lu is compatible with op.T ype then

set lock on lu in appropriate mode on behalf of transaction op.tid ;

if op.Type = W then

DPM(op) {call local DP (M for “master”) with operation}

send “Lock granted” to coordinating TM of transaction

else

put op on a queue for lu

end if

**end case**

...

end switch

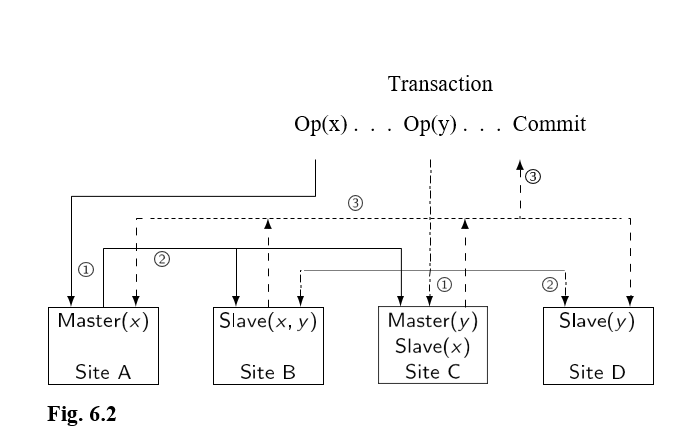
**end**

**6.3.1.3 Primary Copy/Full Replication Transparency**

Thay vì dùng 1 master cho tất cả các mục dữ liệu, ta cho mỗi mục dữ liệu 1 master. Trong trường hợp này, với mỗi mục dữ liệu nhân bản, 1 bản sẽ là bản chính (primary copy). Và vì không có 1 master để xác định thứ tự tuần tự toàn cục, ta cần phải xử lý việc này. Trong tình huống csdl được nhân bản đầy đủ, mỗi bản sao có thể làm bản chính cho 1 mục dữ liệu. Tuy nhiên, nếu chỉ nhân bản 1 phần, phương án limited replication transparency chỉ dùng được nếu cập nhật giao dịch chỉ truy cập các mục dữ liệu có site chính (primary site) ở cùng 1 site. Nếu không, chương trình ứng dụng không thể chuyển tiếp cập nhật giao dịch tới 1 master; nó sẽ phải làm theo kiểu từng thao tác (operation-by-operation), và hơn thế nữa, ta không thể xác định được bản chính nào sẽ đóng vai trò điều phối viên cho 2PC. Bởi vậy, phương án khác là hỗ trợ full transparency, TM cục bộ đóng vai trò điều phối và chuyển tiếp các thao tác tới site chính của mục dữ liệu cần xử lý. Figure 6.2 mô tả chuỗi thao tác. Site A là master của mục dữ liệu x và site B, C chứa bản sao; tương tự Site C là master của mục dữ liệu y và site B, D chứa bản sao.

Phiên bản này chuyển các cập nhật tới các bản sao trong phạm vi giao dịch, yêu cầu kết hợp các kĩ thuật kiểm soát đồng thời. 1 đề xuất là thuật toán primary copy two-phase locking (PC2PL), là 1 phần mở rộng (extension) trực diện của giao thức single master phía trên để xử lý 1 số lỗi. Cơ bản, nó cài đặt các lock manager tại 1 số site và giao trách nhiệm cho mỗi lock manager quản lý các khóa cho 1 số đơn vị khóa của site master. TM sau đó gửi khóa và yêu cầu mở khóa tới lock manager tương ứng. Thuật toán này coi 1 bản sao của mỗi mục dữ liệu như là coi bản chính.

Là sự kết hợp kiểm soát bản sao và kiểm soát đồng thời, tiếp cận bản chính từ các site đòi hỏi 1 phương pháp cầu kì hơn, nhưng nó cũng cải thiện 1 số vấn đề: giảm tải trọng tại site master mà không cần nhiều giao tiếp giữa TM và LM.



(1) Các thao tác đọc hoặc ghi cho mỗi mục dữ liệu được chuyển tới master của mục dữ liệu đó và lệnh ghi đầu tiên phải chuyển tới master.

(2) Sau đó chuyển lệnh ghi tới các bản sao khác.

(3) Các cập nhật có hiệu lực tại thời điểm commit.

**6.3.2 Eager Distributed Protocols**

Có thể cập nhật từ bất cứ đâu, các cập nhật được thực hiện đầu tiên tại bản sao cục bộ, sau đó chuyển tới các bản khác.

Nếu cập nhật bắt nguồn tại 1 site không chứa 1 bản sao của 1 mục dữ thì sẽ được chuyển tiếp tới 1 bản sao ở site khác. Tất cả được thực hiện trong ngữ cảnh cập nhật giao dịch và khi giao dịch được commit, người dùng sẽ nhận thông báo và các cập nhật đã được hoàn thành. Figure 6.3 mô tả chuỗi thao tác cho 1 mục dữ liệu logic x có bản sao tại các site A, B, C, D, và nơi 2 giao dịch cập nhật 2 bản khác nhau (tại site A và D). 1 vấn đề quan trọng là đảm bảo thực hiện các cập nhật mang tính xung đột theo đúng thứ tự ở mọi site mà chúng cùng được thực hiện ( việc thực hiện cục bộ tại các site cũng cần phải tuần tự). Ta có thể dùng các kĩ thuật kiểm soát đồng thời. Thao tác đọc có thể được thực hiện ở bất cứ bản sao nào, nhưng chỉ được ghi trên các bản sao trong phạm vi giao dịch (ví dụ: ROWA) bằng 1 giao thức kiểm soát đồng thời

**6.3.3 Giao thức Lazy Centralized**

Các thuật toán nhân bản lazy centralized tương tự như các thuật toán nhân bản eager centralized ở chỗ các bản cập nhật(update) trước tiên được áp dụng cho một bản sao master (master replica) và sau đó được truyền tới các slave. Sự khác biệt quan trọng là việc lan truyền không diễn ra trong giao dịch cập nhật (update transaction) mà diễn ra sau khi giao dịch commit dưới dạng một giao dịch làm mới(refresh transaction) riêng biệt. Do đó, nếu một trạm(site) slave thực hiện thao tác R(x) trên bản sao cục bộ của nó, nó có thể đọc dữ liệu cũ (không mới), vì x có thể đã được cập nhật tại master, nhưng bản cập nhật có thể chưa được truyền tới các slave.

|  |
| --- |
| Hình 6.3 Các hành động của giao thức nhân bản phân tán eager. (1) Hai thao tác Write(Ghi) được áp dụng trên hai bản sao cục bộ của cùng một mục dữ liệu; (2) Các thao tác Ghi được truyền độc lập đến các bản sao khác; (3) Cập nhật trở thành permanent(vĩnh viễn) tại thời điểm commit (chỉ hiển thị cho Giao dịch 1) |

|  |
| --- |
| Hình 6.4 Các hành động của giao thức nhân bản lazy single master. (1) Bản cập nhật được áp dụng trên bản sao cục bộ; (2) Commit giao dịch làm cho các bản cập nhật trở thành vĩnh viễn tại master; (3) Bản cập nhật được truyền tới các bản sao khác trong các giao dịch làm mới; (4) Giao dịch 2 đọc từ bản sao cục bộ |

**6.3.3.1. Single Master(đơn chủ) với tính minh bạch giới hạn**

Trong trường hợp này, các giao dịch cập nhật được gửi và thực hiện trực tiếp tại trạm master (như trong eager single master); khi giao dịch cập nhật được commit, giao dịch làm mới được gửi đến các slave. Trình tự các bước thực hiện như sau: (1) giao dịch cập nhật trước tiên được áp dụng cho bản sao master(master replica), (2) giao dịch được commit tại master và sau đó (3) giao dịch làm mới được gửi đến các slave (Hình. 6.4).

Khi một trạm slave nhận được R(x), nó sẽ đọc từ bản sao cục bộ của nó và trả về kết quả cho người dùng. Lưu ý rằng, như đã chỉ ra ở trên, bản sao của chính nó có thể không được cập nhật nếu master đang được cập nhật và slave chưa nhận và thực hiện giao dịch làm mới tương ứng. Một W(x) mà một slave nhận được bị reject(từ chối) (và giao dịch bị hủy bỏ), vì W(x) lẽ ra phải được gửi trực tiếp đến master. Khi một slave nhận được một giao dịch làm mới từ master, nó sẽ áp dụng các bản cập nhật cho bản sao cục bộ của nó. Khi nó nhận được Commit hoặc Abort(hủy bỏ) (Việc hủy bỏ chỉ có thể xảy ra đối với các giao dịch chỉ đọc được gửi cục bộ), nó sẽ thực hiện các hành động này cục bộ.

Trường hợp bản sao chính(primary copy) với tính minh bạch giới hạn cũng tương tự nên chúng tôi không bàn chi tiết. Thay vì đi đến trạm single master, W(x) được gửi đến bản sao chính của x; phần còn lại là không phức tạp.

Làm cách nào để đảm bảo rằng các giao dịch làm mới có thể được áp dụng cho tất cả các slave theo cùng 1 thứ tự? Trong kiến trúc này, vì có một bản sao chính single master(a single master copy) cho tất cả các mục dữ liệu, thứ tự có thể được thiết lập đơn giản bằng cách sử dụng mốc thời gian(timestamps). Master sẽ đính kèm một mốc thời gian cho mỗi giao dịch làm mới theo thứ tự commit của giao dịch cập nhật thực tế và các slave sẽ áp dụng các giao dịch làm mới theo thứ tự mốc thời gian.

Một cách tiếp cận tương tự có thể được thực hiện trong bản sao chính(primary copy), trường hợp tính minh bạch giới hạn. Trong trường hợp này, một trạm chứa các bản sao slave(slave copy) của một số mục dữ liệu, khiến trạm đó nhận các giao dịch làm mới từ nhiều master. Việc thực hiện các giao dịch

làm mới này cần phải được sắp xếp theo cùng một cách ở tất cả các slave có liên quan để đảm bảo rằng các trạng thái cơ sở dữ liệu cuối cùng hội tụ. Có một số lựa chọn thay thế sau:

Một giải pháp thay thế là gán các mốc thời gian sao cho các giao dịch làm mới được phát hành từ các master khác nhau có các mốc thời gian khác nhau (bằng cách nối thêm mã định danh trạm vào bộ đếm đơn điệu tại mỗi trạm). Sau đó, các giao dịch làm mới tại mỗi trạm có thể được thực hiện theo thứ tự mốc thời gian của chúng. Tuy nhiên, những giao dịch làm mới không theo thứ tự gây ra khó khăn. Trong các kỹ thuật dựa trên mốc thời gian truyền thống được thảo luận trong Chương. 5, các giao dịch này sẽ bị hủy bỏ; tuy nhiên, trong lazy replication, điều này là không thể vì giao dịch đã được commit tại trạm bản sao chính(primary copy). Khả năng duy nhất là chạy một giao dịch đền bù (thực tế là hủy bỏ giao dịch bằng cách khôi phục các hiệu ứng của nó) hoặc thực hiện đối chiếu cập nhật(update reconciliation) sẽ được thảo luận ngay sau đây. Vấn đề có thể được giải quyết bằng một nghiên cứu cẩn thận hơn về lịch sử kết quả. Một cách tiếp cận là sử dụng cách tiếp cận biểu đồ tuần tự hóa để xây dựng biểu đồ nhân bản có các nút(node) bao gồm các giao dịch (T ) và các trạm (S) và một cạnh <Ti, Sj> tồn tại trong biểu đồ khi và chỉ khi Ti thực hiện Ghi trên bản sao vật lý (đã nhân bản)được lưu trữ tại Sj. Khi một thao tác

(opk) được gửi, các nút (Tk) và các cạnh thích hợp được chèn vào biểu đồ nhân bản, được kiểm tra theo chu trình. Nếu không có chu trình, thì việc thực hiện có thể tiến hành. Nếu một chu trình được phát hiện và nó liên quan đến một giao dịch đã được commit tại master, nhưng các giao dịch làm mới của nó chưa được commit ở tất cả các slave có liên quan, thì giao dịch hiện tại (Tk)

sẽ bị hủy bỏ (sẽ được khởi động lại sau) vì việc thực hiện chu trình sẽ khiến lịch sử là một non-1SR. Mặt khác, Tk có thể đợi cho đến khi các giao dịch khác trong chu trình được hoàn thành (nghĩa là các giao dịch được commit tại các master của chúng và các giao dịch làm mới của chúng được commit tại tất cả các slave). Khi một giao dịch được hoàn thành theo cách này, nút tương ứng và tất cả các cạnh liên thuộc(incident edge) của nó sẽ bị xóa khỏi biểu đồ nhân bản. Giao thức này được chứng minh là tạo ra lịch sử 1SR. Một vấn đề quan trọng là việc duy trì biểu đồ nhân bản. Nếu nó được duy trì bởi một trạm duy nhất(single site), thì đây sẽ trở thành một thuật toán tập trung. Chúng ta coi việc xây dựng và duy trì biểu đồ nhân bản phân tán như một bài tập.

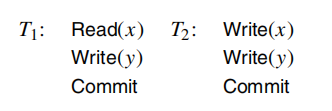
Một cách khác là dựa vào cơ chế truyền thông nhóm(group communication) được cung cấp bởi hạ tầng truyền thông cơ bản(underlying communication infrastructure) (nếu nó có thể đáp ứng). Chúng ta thảo luận về phương pháp thay thế này trong Phần 6.4.

Nhớ lại từ Phần 6.3.1 rằng, trong trường hợp cơ sở dữ liệu được nhân bản một phần, eager primary copy với cách tiếp cận tính minh bạch nhân bản giới hạn(limited replication transparency) sẽ có ý nghĩa nếu các giao dịch cập nhật chỉ truy cập vào các mục dữ liệu có trạm master giống nhau, vì các giao dịch cập nhật được chạy đầy đủ tại một master. Vấn đề tương tự cũng tồn tại trong trường hợp lazy primary copy, cách tiếp cận sao chép hạn chế. Vấn đề phát sinh trong cả hai trường hợp là làm thế nào để thiết kế cơ sở dữ liệu phân tán sao cho các giao dịch có nghĩa có thể được thực hiện. Vấn đề này đã được nghiên cứu trong phạm vi của các lazy protocol và một thuật toán lựa chọn trạm sơ cấp/chính(primary site) đã được đặt ra là, với một tập hợp các giao dịch, một tập hợp các trạm và một tập hợp các mục dữ liệu, việc gán một primary site cho các mục dữ liệu này sao cho tập hợp các giao dịch có thể được thực thi để tạo ra lịch sử toàn cầu 1SR có tồn tại hay không.

**6.3.3.2 Single Master or Primary Copy with Full Replication Transparency (tính minh bạch nhân bản đầy đủ)**

Giờ đây, chúng ta chuyển sang các giải pháp thay thế mang lại tính minh bạch đầy đủ cách cho phép gửi các giao dịch (cả đọc và cập nhật) tại bất kỳ trạm nào và chuyển tiếp các hoạt động của chúng tới single master hoặc tới primary copy thích hợp. Điều này phức tạp và bao gồm đến hai vấn đề: thứ nhất là lịch sử toàn cầu 1SR có thể không được đảm bảo nếu như một giao dịch không cẩn trọng; vấn đề thứ hai là một giao dịch có thể không thấy các bản cập nhật của chính nó. Hai ví dụ sau đây chứng minh những vấn đề này.

*Ví dụ 6.4*: Xem xét kịch bản single master và hai trạm M và B, trong đó M giữ các bản sao master của x và y, và B giữ các bản sao slave của chúng. Bây giờ hãy xem xét hai giao dịch sau: T1 được gửi tại trạm B, trong khi giao dịch T2 được gửi tại trạm M:

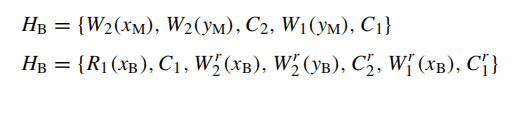


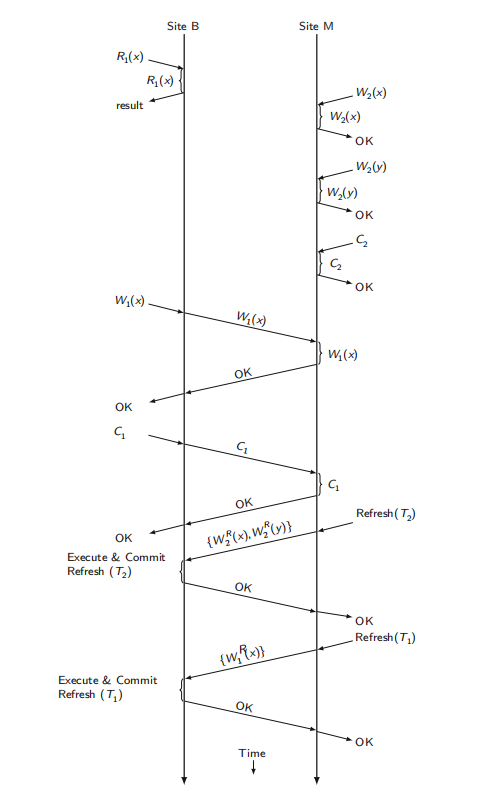
Một cách những điều này sẽ được thực hiện dưới tính minh bạch đầy đủ như sau. T2 sẽ được thực hiện tại trạm M vì nó chứa các bản sao master của cả x và y. Một lúc sau khi nó commit, các giao dịch làm mới cho các hoạt động Ghi của nó được gửi đến trạm B để cập nhật các bản sao slave. Mặt khác, T1 sẽ đọc bản sao cục bộ của x tại trạm B [R1(xB)], nhưng W1(x) của nó sẽ được chuyển tiếp tới bản sao master của x, tại trạm M. Một thời gian sau W1(x) được thực

hiện tại trạm master và commit ở đó, một giao dịch làm mới sẽ được gửi trở lại trạm B để cập nhật bản sao slave. Sau đây là trình tự các bước thực hiện có thể có (Hình 6.5):

1. R1(x) được gửi tại trạm B, nơi nó được thực hiện [R1(xB)];
2. W2(x) được gửi tại trạm M, và nó được thực thi [W2(xM)];
3. W2(y) được gửi tại trạm M, và nó được thực hiện [W2(yB)];
4. T2 gửi Commit của nó tại trạm M và commit tại đó;
5. W1(x) được gửi tại trạm B; vì bản sao master của x ở trạm M, nên Ghi được chuyển tiếp đến M;
6. W1(x) được thực hiện tại trạm M [W1(xM)]; và xác nhận được gửi trở lại trạm B;
7. T1 gửi Commit tại tạm B, commit đó chuyển tiếp đến trạm M; commit được thực hiện ở đó và B được thông báo về commit, T1 cũng commit tại B
8. Trạm M bây giờ gửi giao dịch làm mới cho T2 đến trạm B nơi nó được thực hiện và commit
9. Trạm M cuối cùng gửi giao dịch làm mới cho T1 đến trạm B (đây là bản ghi của T1 đã được thực hiện tại master), nó được thực hiện tại B và commit.

Hai lịch sử sau đây hiện được tạo ra tại hai trạm mà chỉ số trên r trên các hoạt động chỉ ra rằng chúng là một phần của giao dịch làm mới:

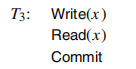




Hình 6.5 Trình tự thời gian thực hiện các giao dịch

Lịch sử toàn cầu thu được trên các mục dữ liệu logic x và y không phải là 1SR.

*Ví dụ 6.5*: Một lần nữa, hãy xem xét một kịch bản single master, trong đó trạm M giữ bản sao master của x và trạm D giữ slave của x. Hãy xem xét giao dịch đơn giản sau:



Theo mô hình thực thi giống như trong Ví dụ 6.4, trình tự các bước sẽ như sau:

1. W3(x) được gửi tại trạm D, chuyển tiếp nó tới trạm M để thực thi;
2. Quá trình Ghi được thực hiện tại M [W3(xM)] và xác nhận được gửi trở lại trạm D;
3. R3(x) được gửi tại trạm D và được thực thi trên bản sao cục bộ [R3(xD)];
4. T3 gửi commit tại D, commit này được chuyển tiếp tới M, được thực hiện tại đó và một thông báo được gửi trở lại trạm D, nơi cũng commit giao dịch;
5. Trạm M gửi một giao dịch làm mới đến trạm D cho hoạt động W3(x);
6. Trạm D thực hiện giao dịch làm mới và commit nó.

Lưu ý rằng, do giao dịch làm mới được gửi đến trạm D sau khi T3 committại trạm M, nên ở bước 3 khi nó đọc giá trị của x tại trạm D, nó sẽ đọc giá trị cũ và không thấy giá trị Write của chính nó ngay trước khi Read.

Vì những vấn đề này, không có quá nhiều đề xuất về tính minh bạch đầy đủ trong các thuật toán lazy replication. Một ngoại lệ đáng chú ý là một thuật toán xem xét trường hợp single master và cung cấp một phương pháp để kiểm tra tính hợp lệ của single master, tại điểm commit, tương tự như optimistic concurrency control(kiểm soát đồng thời lạc quan). Ý tưởng cơ bản là như sau. Xét một giao dịch T ghi một mục dữ liệu x. Tại thời điểm commit của giao dịch T, master tạo mốc thời gian cho T và sử dụng mốc thời gian này để đặt mốc thời gian cho bản sao master (xM) cái mà ghi lại mốc thời gian của giao dịch cuối cùng cập nhật nó (last\_modified(xM)). Mốc thời gian cũng được thêm vào các giao dịch làm mới. Khi các giao dịch làm mới được nhận tại các slave, slave cũng thiết đặt các bản sao của chúng thành cùng giá trị này, tức là last\_modified(xi) <- last\_modified(xM).

Việc tạo dấu thời gian cho T tại master tuân theo quy tắc sau:

Mốc thời gian cho giao dịch T phải lớn hơn tất cả các mốc thời gian đã phát hành trước đó và phải nhỏ hơn mốc thời gian last\_modified của các mục dữ liệu mà nó đã truy cập. Nếu không thể tạo mốc thời gian như vậy, thì T bị hủy bỏ.

Việc kiểm tra này đảm bảo rằng hoạt động đọc đọc các giá trị chính xác. Ví dụ: trong Ví dụ 6.4 trạm master M sẽ không thể gán mốc thời gian thích hợp

cho giao dịch T1 khi nó được commit, vì last\_modified(xM) sẽ phản ánh bản cập nhật được thực hiện bởi T2. Do đó, T1 sẽ bị hủy bỏ.

Mặc dù thuật toán này xử lý vấn đề đầu tiên mà chúng ta đã thảo luận ở trên, nhưng nó không tự động xử lý vấn đề giao dịch không nhìn thấy bản ghi của chính nó (cái mà chúng tôi gọi là đảo ngược giao dịch trước đó). Để giải quyết vấn đề này, chúng tôi đề xuất rằng nên duy trì một danh sách tất cả các bản cập nhật mà một giao dịch thực hiện và danh sách này được tham khảo khi một thao tác Đọc được thực thi. Tuy nhiên, vì chỉ master biết các bản cập nhật, nên danh sách phải được duy trì ở master và tất cả các hoạt động Đọc và Ghi phải được thực hiện ở master.

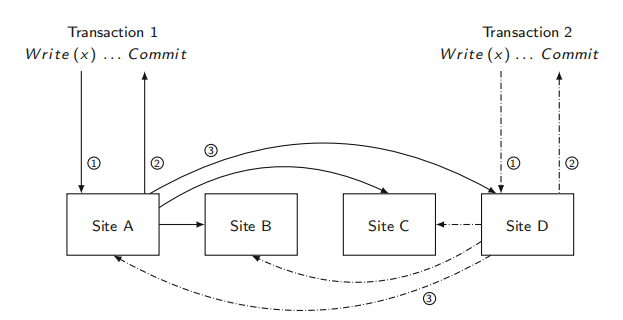
**6.3.4. Lazy Distributed Protocols**

Các giao thức nhân bản lazy distributed là những giao thức phức tạp nhất do thực tế là các bản cập nhật có thể xảy ra trên bất kỳ bản sao nào và chúng được truyền đến các bản sao khác một cách lười nhác(lazy) (Hình 6.6).

Hoạt động của giao thức tại trạm nơi giao dịch được gửi rất đơn giản: cả thao tác Đọc và Ghi đều được thực hiện trên bản sao cục bộ và giao dịch được xác nhận cục bộ. Đôi khi sau khi commit, các bản cập nhật được truyền đến các trạm khác bằng các giao dịch làm mới.

Sự phức tạp nảy sinh trong quá trình xử lý các bản cập nhật này tại các trạm khác. Khi các giao dịch làm mới đến một trạm, chúng cần được lên lịch cục bộ, được thực hiện bởi cơ chế kiểm soát tương tranh cục bộ. Có thể thực hiện tuần tự hóa thích hợp các giao dịch làm mới này bằng cách sử dụng các kỹ thuật đã thảo luận trong các phần trước. Tuy nhiên, nhiều giao dịch có thể cập nhật

đồng thời các bản sao khác nhau của cùng một mục dữ liệu tại các trạm khác nhau và những cập nhật này có thể xung đột với nhau. Những thay đổi này cần phải được đối chiếu(reconcile) và điều này làm phức tạp thứ tự làm mới giao dịch. Dựa trên kết quả đối chiếu, thứ tự thực hiện các giao dịch làm mới được xác định và các bản cập nhật được áp dụng tại mỗi trạm.



Hình 6.6 Các hành động của giao thức nhân bản lazy distributed. (1) Hai bản cập nhật được áp dụng trên hai bản sao cục bộ; (2) Commit giao dịch làm cho các bản cập nhật trở nên vĩnh viễn; (3) Các bản cập nhật được truyền độc lập đến các bản sao khác

Vấn đề quan trọng ở đây là đối chiếu. Người ta có thể thiết kế một thuật toán đối chiếu mục đích chung dựa trên kinh nghiệm. Ví dụ: các bản cập nhật có thể được áp dụng theo thứ tự mốc thời gian (nghĩa là những bản cập nhật có mốc thời gian sau sẽ luôn thắng) hoặc người ta có thể ưu tiên các bản cập nhật bắt nguồn từ các trạm nhất định (có thể có các trạm quan trọng hơn). Tuy nhiên, đây là những phương pháp đặc biệt và việc đối chiếu thực sự phụ thuộc vào ngữ

nghĩa của ứng dụng. Hơn nữa, bất kể kỹ thuật đối chiếu nào được sử dụng, một số cập nhật sẽ bị mất. Lưu ý rằng thứ tự dựa trên mốc thời gian sẽ chỉ hoạt động nếu mốc thời gian dựa trên đồng hồ cục bộ được đồng bộ hóa. Như chúng ta đã thảo luận trước đó, điều này khó đạt được trong các hệ thống phân tán quy mô lớn. Cách tiếp cận dựa trên mốc thời gian đơn giản, kết hợp một số trạm và đồng hồ cục bộ, đưa ra tùy chọn tùy ý giữa các giao dịch có thể không có cơ sở thực trong logic ứng dụng. Lý do mốc thời gian hoạt động tốt trong kiểm soát đồng thời và không hoạt động trong trường hợp này là vì trong kiểm soát đồng thời, chúng ta chỉ quan tâm đến việc xác định *một số* thứ tự; ở đây chúng ta quan tâm đến việc xác định một thứ tự *cụ thể* phù hợp với ngữ nghĩa của ứng dụng.

**6.4. Truyền thông nhóm**

Như đã thảo luận trong phần trước, chi phí hoạt động của các giao thức nhân bản có thể cao—đặc biệt xét về tổng chi phí thông điệp(message overhead). Một mô hình chi phí rất đơn giản cho các thuật toán nhân bản như sau. Nếu có *n* bản sao và mỗi giao dịch bao gồm m thao tác cập nhật, thì mỗi giao dịch phát sinh *n\*m* thông điệp (nếu có thể phát đa hướng, *m* thông điệp là đủ). Nếu hệ thống muốn duy trì thông lượng là *k* giao dịch mỗi giây, điều này gây ra *k\*n\*m* thông điệp mỗi giây (hoặc *k\*m* trong trường hợp phát đa hướng). Người ta có thể thêm sự phức tạp cho hàm chi phí này bằng cách xem xét thời gian thực hiện của từng thao tác (có lẽ dựa trên tải hệ thống) để có được hàm chi phí theo thời gian. Vấn đề với nhiều giao thức nhân bản đã thảo luận ở trên (đặc biệt là các giao thức phân tán) là chi phí thông điệp của chúng cao.

Một vấn đề quan trọng trong việc triển khai hiệu quả các giao thức này là giảm chi phí thông điệp. Các giải pháp đã được đề xuất sử dụng các giao thức truyền thông nhóm cùng với các kỹ thuật phi truyền thống để xử lý các giao dịch cục bộ. Các giải pháp này đưa ra hai sửa đổi: chúng không sử dụng 2PC tại thời điểm commit mà dựa vào các giao thức truyền thông nhóm cơ bản để đảm bảo sự đồng thuận, và chúng sử dụng lan truyền cập nhật bị trì hoãn thay vì lan truyền cập nhật đồng bộ.

Đầu tiên chúng ta hãy xem lại ý tưởng truyền thông. Một hệ thống truyền thông nhóm cho phép một nút phát đa hướng một thông điệp tới tất cả các nút của một nhóm với sự đảm bảo gửi, nghĩa là thông điệp cuối cùng sẽ được truyền/phát tới tất cả các nút. Hơn nữa, nó có thể cung cấp các nguyên mẫu phát đa hướng với các thứ tự truyền tải/phát (delivery order) khác nhau, chỉ một trong số đó là quan trọng đối với cuộc thảo luận của chúng ta: tổng chi phí. Trong phát đa hướng theo thứ tự tổng thể(total ordered multicast), tất cả các thông điệp được gửi bởi các nút khác nhau được truyền/phát đi theo cùng một thứ tự tại tất cả các nút. Điều này rất quan trọng trong việc hiểu các cuộc thảo luận sau đây.

Chúng ta sẽ làm rõ việc sử dụng truyền thông nhóm bằng cách xem xét hai giao thức. Cái đầu tiên là một giao thức eager distributed thay thế, trong khi cái thứ hai là một giao thức lazy centralized.

Giao thức eager distributed dựa trên truyền thông nhóm sử dụng chiến lược xử lý cục bộ trong đó thao tác Ghi được thực hiện trên các bản sao ẩn cục bộ(local shadow copy) nơi giao dịch được gửi và sử dụng truyền thông nhóm theo thứ tự tổng thể để phát đa hướng tập hợp các thao tác ghi của giao dịch tới tất cả các trạm bản sao(replica site). Truyền thông theo thứ tự tổng thể đảm bảo rằng tất cả các trạm đều nhận được các thao tác ghi theo đúng thứ tự, do đó đảm bảo thứ tự tuần tự hóa đồng nhất tại mọi trạm. Để đơn giản hóa việc mô tả, trong cuộc thảo luận sau đây, chúng tôi giả định rằng cơ sở dữ liệu được nhân bản đầy đủ và mỗi trạm thực hiện thuật toán kiểm soát tương tranh 2PL.

Giao thức thực hiện một giao dịch Ti theo bốn bước (hành động kiểm soát tương tranh cục bộ không được chỉ định):

**I. Giai đoạn/Pha xử lý cục bộ.** Một hoạt động Ri(x) được thực hiện tại trạm nơi nó được gửi (đây là trạm master cho giao dịch này). Thao tác Wi(x) cũng được thực hiện tại trạm master, nhưng trên một bản sao ẩn (xem chương trước để thảo luận về phân trang ẩn).

**II. Giai đoạn/Pha truyền thông.** Nếu Ti chỉ bao gồm các thao tác Đọc, thì nó có thể được commit tại master. Nếu nó bao gồm các thao tác Ghi (nghĩa là nếu nó là một giao dịch cập nhật), thì TM tại trạm master của Ti (nghĩa là trạm mà Ti được gửi) tập hợp các bản ghi thành một thông điệp ghi WMi và phát đa hướng nó tới tất cả các trạm bản sao (bao gồm chính nó) sử dụng truyền thông nhóm theo thứ tự tổng thể.

**III. Giai đoạn/Pha khóa**. Khi WMi được truyền/phát đi tại một trạm Sj , nó yêu cầu tất cả các khóa trong WMi trong một bước nguyên tử. Điều này có thể được thực hiện bằng cách lấy một chốt (dạng khóa nhẹ hơn) trên bảng khóa được giữ cho đến khi tất cả các khóa được cấp hoặc các yêu cầu được xử lý. Các hành động sau đây được thực hiện:

**1.** Đối với mỗi W(x) trong WMi (cho xj chỉ bản sao của x tồn tại tại trạm Sj):

**(a)** Nếu không có giao dịch nào khác khóa xj , thì khóa ghi trên xj đã được cấp. **(b)** Mặt khác, thử nghiệm xung đột được thực hiện:

• Nếu có một giao dịch cục bộ Tk đã khóa xj, nhưng đang trong giai đoạn đọc hoặc giao tiếp cục bộ, thì Tk bị hủy bỏ. Hơn nữa, nếu Tk đang trong giai đoạn truyền thông, thông điệp quyết định cuối cùng “hủy bỏ” sẽ được phát đa hướng tới tất cả các trạm. Ở giai đoạn này, các xung đột đọc/ ghi được phát hiện và các giao dịch đọc cục bộ bị hủy bỏ một cách đơn giản. Lưu ý rằng chỉ các thao tác đọc cục bộ mới nhận được khóa trong suốt giai đoạn thực thi cục bộ, vì ghi cục bộ chỉ được thực hiện trên các bản sao ẩn. Do đó, không cần kiểm tra ghi/xung đột ghi ở giai đoạn này.

• Mặt khác, yêu cầu khóa Wi(xj ) được đưa vào hàng đợi cho xj .

**2.** Nếu Ti là một giao dịch cục bộ (hãy nhớ rằng thông điệp cũng được gửi đến trạm nơi Ti bắt đầu, trong trường hợp j = i), thì trạm đó có thể commit giao dịch, do đó, nó sẽ phát đa hướng một thông điệp “commit”. Lưu ý rằng thông điệp commit gửi ngay khi khóa được yêu cầu chứ không phải sau khi ghi; do đó, đây không phải là thực thi 2PC.

**IV. Giai đoạn ghi**. Khi một trạm có thể nhận được khóa ghi, nó sẽ áp dụng bản cập nhật tương ứng (đối với trạm master, điều này có nghĩa là bản sao ẩn được tạo thành phiên bản hợp lệ). TRạm nơi Ti được gửi có thể commit và giải phóng tất cả các khóa. Các trạm khác phải đợi thông điệp quyết định và chấm dứt cho phù hợp.

Lưu ý rằng trong giao thức này, điều quan trọng là đảm bảo rằng các giai đoạn khóa của các giao dịch đồng thời được thực hiện theo cùng một thứ tự tại mỗi trạm; đó là những gì phát đa hướng theo thứ tự tổng thể đạt được. Cũng lưu ý rằng không có yêu cầu thứ tự trên các thông điệp quyết định (bước III.2) và chúng có thể được gửi theo bất kỳ thứ tự nào, ngay cả trước khi gửi WM tương ứng. Nếu điều này xảy ra, thì các trạm nhận được thông điệp quyết định trước WM chỉ cần đăng ký quyết định nhưng không thực hiện bất kỳ hành động nào. Khi thông điệp WM đến, các trạm có thể thực hiện các giai đoạn khóa và ghi và chấm dứt giao dịch theo thông điệp quyết định đã truyền/phát trước đó.

Giao thức này tốt hơn đáng kể về mặt hiệu suất so với giao thức ngây thơ(naive) được thảo luận trong Phần. 6.3.2. Đối với mỗi giao dịch, master gửi hai thông điệp: một khi nó gửi WM và một khi nó truyền thông quyết định. Do đó, nếu chúng ta muốn duy trì thông lượng hệ thống là *k* giao dịch mỗi giây, thì tổng số thông điệp là *2k* chứ không phải *k\*m*, như trường hợp của naive protocol (giả sử phát đa hướng trong cả hai trường hợp). Hơn nữa, hiệu suất hệ thống được cải thiện bằng cách sử dụng lan truyền hăng hái(eager propagation) bị trì hoãn do việc đồng bộ hóa giữa các trạm bản sao cho tất cả các thao tác Ghi được thực hiện một lần ở cuối thay vì trong suốt quá trình thực hiện giao dịch.

Ví dụ thứ hai về việc sử dụng truyền thông nhóm mà chúng ta sẽ thảo luận là trong phạm vi các thuật toán lazy centralized. Nhớ lại rằng một vấn đề quan trọng trong trường hợp này là đảm bảo rằng các giao dịch làm mới được sắp xếp theo cùng một cách ở tất cả các slave có liên quan để các trạng thái cơ sở dữ liệu hội tụ. Nếu có sẵn phát đa hướng theo thứ tự tổng thể, các giao dịch làm mới được gửi bởi các trạm master khác nhau sẽ được truyền/phát theo cùng một thứ tự ở tất cả các slave. Tuy nhiên, phát đa hướng theo thứ tự toàn thể có

chi phí thông điệp cao, điều này có thể hạn chế khả năng mở rộng của nó. Có thể nới lỏng yêu cầu thứ tự của hệ thống truyền thông và để giao thức nhân bản chịu trách nhiệm sắp thứ tự thực hiện các giao dịch làm mới. Chúng tôi sẽ làm rõ sự thay thế này bằng một giao thức giả định truyền thông đa hướng theo thứ tự FIFO với độ trễ giới hạn cho truyền thông (gọi nó là *Max*), và giả định rằng các đồng hồ được đồng bộ hóa một cách lỏng lẻo để chúng chỉ có thể không đồng bộ tới mức  . Giả định có một chức năng quản lý giao dịch thích hợp tại mỗi trạm. Kết quả của giao thức nhân bản tại mỗi slave phải duy trì một “running queue(hàng đợi đang chạy)" chứa danh sách các giao dịch làm mới theo thứ tự, đây là đầu vào cho trình quản lý giao dịch để thực thi cục bộ. Do đó, giao thức đảm bảo rằng các thứ tự trong running queue tại mỗi trạm slave nơi một tập hợp các giao dịch làm mới chạy là như nhau.

Tại mỗi trạm slave, một "pending queue(hàng đợi đang chờ xử lý)" được duy trì cho mỗi trạm master của trạm slave này (nghĩa là nếu trạm slave có các bản sao của x và y có các trạm master lần lượt là S1 và S2, thì có hai pending queue, q1 và q2, lần lượt tương ứng với trạm S1 và S2). Khi một giao dịch làm mới được tạo ở trạm master , nó được gán một mốc thời gian ts(RTi) tương ứng với giá trị thời gian thực tại thời điểm commit của giao dịch cập nhật Ti tương ứng. Khi RTi đến một slave, nó được đưa vào hàng đợi qk. Tại mỗi thông điệp đến, các phần tử trên cùng của tất cả các pending queue được quét và phần tử có mốc thời gian thấp nhất được chọn làm RT mới (new\_RT ) sẽ được xử lý. Nếu new\_RT đã thay đổi kể từ chu trình trước (nghĩa là RT mới đến với mốc thời gian thấp hơn so với giá trị được chọn trong chu trình trước), thì RT có mốc thời gian thấp hơn sẽ trở thành new\_RT và được xem xét để lên lịch.

Khi một giao dịch làm mới được chọn là new\_RT, nó không được đưa ngay vào “running queue” cho người quản lý giao dịch; việc lên lịch cho một giao dịch làm mới có tính đến độ trễ tối đa và độ lệch khả dĩ(possible drift) trong đồng hồ cục bộ. Điều này được thực hiện để đảm bảo rằng bất kỳ giao dịch làm mới nào có thể bị trì hoãn đều có cơ hội đến được với slave. Thời điểm một RTi được đưa vào “running queue” tại một trạm slave là delivery\_time = ts(new\_RT )+*Max* +. Do hệ thống truyền thông đảm bảo giới hạn trên của *Max* cho việc phát thông điệp và do độ lệch tối đa trong đồng hồ cục bộ (xác định mốc thời gian) nên giao dịch làm mới không thể bị trì hoãn nhiều hơn thời gian delivery\_time trước khi đến được tất cả các slave dự tính. Do đó, giao thức đảm bảo rằng một giao dịch làm mới được lên lịch để thực hiện tại một slave khi thỏa mãn các điều kiện sau: (1) tất cả các thao tác ghi của giao dịch cập nhật tương ứng được thực hiện tại master, (2) theo thứ tự xác định bởi mốc thời gian của giao dịch làm mới (phản ánh thứ tự commit của giao dịch cập nhật) và (3) sớm nhất tại thời gian thực tương đương với thời gian delivery\_time của nó. Điều này đảm bảo rằng các bản cập nhật trên các bản sao phụ(secondary copy) tại các trạm slave tuân theo cùng một thứ tự thời gian mà các bản sao chính(primary copy) của chúng đã được cập nhật và thứ tự này sẽ giống nhau ở tất cả các slave có liên quan, giả sử rằng hạ tầng truyền thông cơ bản có thể đảm bảo *Max* và . Đây là một ví dụ về thuật toán lười nhác(lazy algorithm) đảm bảo lịch sử toàn cầu 1SR, nhưng tính nhất quán lẫn nhau yếu cho phép các giá trị bản sao phân kỳ theo một khoảng thời gian định trước.

* full transparency: tính minh bạch đầy đủ
* underlying communication infrastructure: hạ tầng truyền thông cơ bản
* total ordered multicast: phát đa hướng theo thứ tự tổng thể/phát đa hướng tổng thể theo thứ tự/phát đa hướng sắp đặt toàn phần

**6.5. Lỗi và Nhân bản Lazy**

**Mở đầu:**

Cho đến thời điểm này, chúng tôi đã tập trung vào các giao thức nhân bản mà không có bất kỳ lỗi nào. Vậy nếu có lỗi hệ thống thì tính nhất quán lẫn nhau sẽ như thế nào?

Việc xử lý các lỗi khác nhau giữa 2 phương pháp nhân bản Lazy và nhân bản Eager

Cách duy trì tính nhất quán của bản sao khi xảy ra lỗi:

* + Lỗi trang web: Read-One-Write-All-Available
  + Lỗi giao tiếp: Biểu quyết
  + Phân vùng mạng: Biểu quyết

1. **Lỗi và nhân bản Lazy**

Trường hợp này tương đối đơn giản vì các giao thức nhân bản Lazy cho phép sự khác nhau giữa bản master copy và bản replica

Do đó, khi lỗi giao tiếp làm cho một hoặc nhiều trang web không thể truy cập được (phần sau do phân vùng mạng), các trang web khả dụng có thể tiếp tục xử lý.

Trong trường hợp phân vùng mạng, người ta có thể cho phép các hoạt động tiến hành độc lập trong nhiều phân vùng và sau đó lo ngại về sự hội tụ của các trạng thái cơ sở dữ liệu khi sửa chữa bằng cách sử dụng các kỹ thuật giải quyết xung đột (phần 6.3.4)

Trước khi hợp nhất, cơ sở dữ liệu ở nhiều phân vùng khác nhau, nhưng chúng được đối chiếu tại thời điểm hợp nhất.

1. **Lỗi và nhân bản Eager**

Tất cả các kỹ thuật Eager đều triển khai một số loại giao thức ROWA(Read one write all), đảm bảo rằng khi giao tác cập nhật được thực hiện, tất cả các bản sao đều có cùng giá trị

Nhược điểm của ROWA: Ngay cả khi một trong các bản sao không có sẵn, giao tác cập nhật cũng không thể bị chấm dứt. => ROWA không đáp ứng được một trong những mục đích cơ bản của nhân bản, cụ thể là cung cấp tính khả dụng cao hơn.

=> Một giải pháp thay thế cho ROWA, giải quyết vấn đề về tính khả dụng thấp, là giao thức Read-One/Write-All Available (ROWA-A). Ý tưởng chung là các lệnh ghi được thực hiện trên tất cả các bản sao có sẵn và giao tác kết thúc. Các bản sao không có sẵn vào thời điểm đó sẽ phải "bắt kịp" khi chúng có sẵn.

2 phiên bản của ROWAA:

* *Giao thức bản sao có sẵn:*
  + Giao tác cập nhật: Ti
  + Điều phối viên gửi Wi(x) đến tất cả các trang web slave (nơi chứa các bản sao của x)
  + Nếu trả lời: Tiếp tục cập nhật
  + Nếu hết thời gian mà không trả lời (missing write):
    - Coi những trang web này là không khả dụng
    - Các trang web này cập nhật CSDL lên trạng thái mới nhất khi chúng gửi lại
    - Lưu ý: Các trang web này có thể không biết về sự tồn tại của Ti và bản cập nhật cho x mà Ti đã thực hiện nếu chúng không khả dụng trước khi Ti bắt đầu
  + Có 2 vấn đề:
    - Có thể các trang web mà điều phối viên nghĩ là không khả dụng trên thực tế đã được thiết lập và đang chạy và có thể đã cập nhật x nhưng xác nhận của chúng có thể chưa đến được với điều phối viên trước khi hết giờ
    - Một số trang web này có thể không khả dụng khi Ti bắt đầu và có thể đã khôi phục kể từ đó và đã bắt đầu thực hiện các giao tác
  + Do đó, điều phối viên phải thực hiện quy trình xác nhận trước khi thực hiện Ti:
    - 1. Kiểm tra cả các trang web mà nó cho là không khả dụng vẫn không khả dụng không. Thực hiện điều này bằng cách gửi một tin nhắn yêu cầu tới mọi trang web này. Những trang web khả dụng sẽ trả lời. Nếu điều phối viên nhận được phản hồi từ một trong những trang web này, thì sẽ hủy bỏ Ti vì nó không biết trạng thái của trang trước đó

(có thể trang web đó đã sẵn sàng và đã thực hiện Wi(x) ban đầu nhưng xác nhận của nó đã bị trì hoãn hoặc nó thực sự không có sẵn khi Ti bắt đầu nhưng sau đó lại có sẵn và thậm chí có thể thực hiện Wj (x) thay mặt cho một giao tác khác Tj) Trong trường hợp sau, tiếp tục với Ti sẽ làm cho lịch trình thực thi không thể tuần tự hóa được.

* + - 2. Nếu điều phối viên của T không nhận được bất kỳ phản hồi nào từ bất kỳ trang web nào mà nó cho là không khả dụng, thì sẽ kiểm tra để đảm bảo rằng tất cả các trang web khả dụng khi Wi(x) thực thi vẫn khả dụng. Nếu có, thì T có thể tiến hành commit. Bước này có thể được tích hợp vào một giao thức thực hiện
* *Giao thức ROWAA phân tán:*
  + Mỗi trang web S có một tập V các trang web mà nó tin rằng có sẵn (đây là “view” mà S có về cấu hình hệ thống)
  + Cụ thể, khi một giao tác Ti được gửi, tầm nhìn của điều phối viên phản ánh tất cả các trang web mà điều phối viên biết là có sẵn (VC(Ti))
  + Ri(x): được thực hiện trong bất kỳ bản replica trong VC(Ti)
  + Wi(x): cập nhật tất cả các bản copy trong VC(Ti)
  + Điều phối viên cần thực hiện:
    - Đọc bất kỳ bản sao nào trong V; cập nhật tất cả các bản sao trong V
    - Nếu khi kết thúc giao tác, chế độ xem đã thay đổi, giao tác được hủy bỏ
  + Tất cả các trang web phải có cùng 1 chế độ xem
  + Sửa đổi V:
    - Giao tác nguyên tử đặc biệt được chạy ở tất cả các trang web, đảm bảo không có chế độ xem nào đồng thời được tạo
    - Chỉ định thời điểm cho mỗi V khi nó được tạo, trang web chỉ chấp nhận chế độ xem mới nếu số phiên bản cao hơn số hiện tại
  + Khôi phục: Nhận các bản cập nhật bị bỏ lỡ từ bất kỳ nút đang hoạt động nào. Vấn đề: Không có chuỗi giao tác duy nhất
* *Giao thức dựa trên biểu quyết*
  + Mỗi thao tác đọc và ghi phải có đủ số phiếu bầu để có thể thực hiện
  + Có thể: bi quan và lạc quan, nhưng chúng ta chỉ xét trường hợp bi quan
  + Thuật toán bỏ phiếu sớm nhất (Thuật toán Thomas): hoạt động trên CSDL được sao chép đầy đủ
    - Chỉ định 1 phiếu bầu cho 1 bản sao (Vi) sao cho ∑iVi=V
    - Mỗi thao tác phải có số đại biểu đọc (Vr) hoặc số đại biểu ghi(Vw) để đọc/ghi 1 mục dữ liệu tương ứng
    - Sau đó, các quy tắc sau phải được tuân theo khi xác định số đại biểu:
      * Vr+Vw>V (quy tắc ngưỡng):

Một mục dữ liệu không được đọc và ghi bởi 2 giao tác đồng thời, tránh xung đột đọc ghi

* + - * Vw>V/2 (quy tắc phân bổ):

Đảm bảo 2 thao tác đọc/ghi từ 2 giao tác không thể xảy ra đồng thời trên cùng 1 mục dữ liệu

* Đảm bảo tính tương đương 1 bản sao và khả năng tuần tự hóa được duy trì
* Đảm bảo rằng 2 giao tác được bắt đầu ở hai phân vùng khác nhau và truy cập cùng một dữ liệu không thể kết thúc cùng một lúc
  + Trong trường hợp phân vùng mạng, giao thức này hoạt động tốt, vì chúng xác định giao tác nào sẽ kết thúc dựa trên phiếu bầu nhận được
  + Khó khăn với phiên bản giao thức này:
    - Các giao tác phạt đạt được số đại biểu cần thiết để đọc dữ liệu
    - => Làm chậm và không cần thiết quyền truy cập đọc vào CSDL
  + Khắc phục: Một giao thức bỏ phiếu dựa trên số đại biểu khắc phục được nhược điểm này
    - Đưa ra giả thuyết nhất định về tầng giao tiếp cơ bản và sự xuất hiện của các lỗi, những lỗi “clean”:
      * 1. Các lỗi làm thay đổi cấu trúc liên kết của mạng được các trang phát hiện ngay lập tức
      * 2. Mỗi trang web có 1 chế độ xem mạng có những trang web nào mà chúng có thể giao tiếp
    - Nếu có 1 mạng truyền thông đảm bảo 2 điều kiện trên thì giao thức kiểm soát bản sao đơn giản được triển khai từ ROWA-A: Đầu tiên kiểm tra xem các trang có nằm trong cùng 1 phân vùng với trang mà giao thức đang hoạt động không? Sau đó thực hiện quy tắc ROWA: đọc bất kỳ bản sao nào của mục dữ liệu và ghi tất cả các bản sao vào phân vùng đó
    - Lưu ý: thao tác đọc hoặc ghi sẽ chỉ thực hiện trong một phân vùng. Do đó, đây là một giao thức bi quan đảm bảo khả năng tuần tự hóa một bản sao, nhưng chỉ trong phân vùng đó. Khi phân vùng sửa chữa, cơ sở dữ liệu được khôi phục bằng cách truyền kết quả cập nhật sang các phân vùng khác.
    - Nhược điểm giao thức này: Giả định về các lỗi không thực tế vì hầu hết các lỗi mạng đều không “clean” tức là vi phạm 2 quy tắc trên và có hậu quả nặng nề với giao thức kiểm soát bản sao và khả năng duy trì tính tuần tự của bản sao

(Giải thích: Có một khoảng thời gian trễ giữa sự xuất hiện của một lỗi và một trang web phát hiện ra nó. Do sự chậm trễ này, một trang web có thể nghĩ rằng nó nằm trong một phân vùng trong khi thực tế các lỗi tiếp theo đã đặt nó vào một phân vùng khác. Hơn nữa, độ trễ này có thể khác nhau đối với các trang web khác nhau. Do đó, hai trang web trong cùng một phân vùng nhưng hiện đang ở trong các phân vùng khác nhau có thể tiếp tục trong một thời gian với giả định rằng chúng vẫn ở trong cùng một phân vùng)

* + - => Giải pháp: Xây dựng trên lớp giao tiếp vật lý 1 lớp trừu tượng khác che giấu các đặc điểm các lỗi không “clean” của lớp giao tiếp vật lý => Trình bày cho giao thức điều khiển bản sao đay là lỗi “clean”

Ngoài ra lớp trừu tượng này cung các phân vùng ảo

* + - Ưu điểm giao thức này: Đơn giản, không phát sinh chi phí nào để duy trì số đại biểu cần thiết cho việc truy cập đọc=> Nhanh như mạng không phân vùng. Ngoài ra nó tổng quát để giao thức kiểm soát bản sao không cần phân biệt lỗi trang web và lỗi phân vùng mạng

**6.6. Tổng kết**

Trong chương này, húng ta đã nói về các cách tiếp cận khác nhau để nhân bản dữ liệu và trình bày các giao thức phù hợp trong các trường hợp khác nhau. Mỗi giao thức đều có những ưu điểm và nhược điểm của chúng.

Các giao thức Eager Centralized đơn giản để triển khai, chúng không yêu cầu phối hợp cập nhật giữa các trang web và chúng được đảm bảo các lịch sử tuần tự hóa một bản sao. Tuy nhiên chúng khiến các trang web chính khá nặng, có khả năng khiến chúng trở thành nút cổ chai. Do đó, chúng khó mở rộng quy mô hơn, đặc biệt là trong kiến trúc trang chủ duy nhất—các phiên bản sao chép chính có các thuộc tính khả năng mở rộng tốt hơn do các trách nhiệm chính được phân bổ phần nào. Các giao thức này dẫn đến thời gian phản hồi lâu (dài nhất trong số bốn lựa chọn thay thế), vì quyền truy cập vào bất kỳ dữ liệu nào phải đợi cho đến khi cam kết của bất kỳ giao tác nào hiện đang cập nhật dữ liệu đó (sử dụng 2PC, rất tốn kém). Hơn nữa, các bản sao cục bộ được sử dụng ít, chỉ dành cho các thao tác đọc. Do đó, nếu khối lượng công việc đòi hỏi nhiều cập nhật, các giao thức tập trung háo hức có khả năng bị kém hiệu suất.

Các giao thức Lazy Centralized có thời gian phản hồi rất ngắn do các giao tác thực hiện và cam kết tại máy chủ và không cần đợi các trang web phụ hoàn thành. Cũng không cần phối hợp giữa các trang trong quá trình thực hiện giao tác cập nhật, do đó giảm số lượng thông báo. Mặt khác, tính nhất quán lẫn nhau (tức là độ mới của dữ liệu ở tất cả các bản sao) không được đảm bảo vì các bản sao cục bộ có thể bị lỗi thời. Điều này có nghĩa là không thể đọc cục bộ và đảm bảo rằng bản sao cập nhật nhất đã được đọc

Cuối cùng, các giao thức multimaster lazy có thời gian phản hồi ngắn nhất và tính khả dụng cao nhất. Điều này là do mỗi giao tác được thực hiện cục bộ, không có sự phối hợp phân tán. Chỉ sau khi chúng thực hiện, các bản sao khác mới được cập nhật thông qua các giao tác làm mới. Tuy nhiên, đây cũng là thiếu sót của các giao thức này—các bản sao khác nhau có thể được cập nhật bởi các giao tác khác nhau, yêu cầu các giao thức đối chiếu phức tạp và dẫn đến các bản cập nhật bị mất.

Nhân bản đã được nghiên cứu rộng rãi trong cộng đồng điện toán phân tán cũng như cộng đồng cơ sở dữ liệu. Mặc dù có những điểm tương đồng đáng kể trong cách xác định vấn đề ở hai môi trường, nhưng cũng có những khác biệt quan trọng. Có lẽ hai sự khác biệt quan trọng hơn là như sau.

Nhân bản dữ liệu tập trung vào dữ liệu, trong khi nhân bản tính toán cũng quan trọng không kém trong điện toán phân tán. Đặc biệt, những lo ngại về nhân bản dữ liệu trong môi trường di động liên quan đến hoạt động bị ngắt kết nối đã nhận được sự quan tâm đáng kể. Thứ hai, tính nhất quán của cơ sở dữ liệu và giao tác là hết sức quan trọng trong nhân bản dữ liệu; trong điện toán phân tán, mối quan tâm về tính nhất quán không cao trong danh sách ưu tiên. Do đó, tiêu chí nhất quán yếu hơn đáng kể đã được khẳng định.

Note:

Copy: bản sao

Replica: bản sao một cách chính xác

Commit: thực hiện