CHƯƠNG VI
HỆ THỐNG PHỤC HỒI
(Recovery system)

MỤC ĐÍCH

Một hệ thống máy tính, cũng giống như các thiết bị cơ - điện khác, luôn có nguy cơ bị hỏng hóc do nhiều nguyên nhân hư đĩa, mất nguồn, lỗi phần mềm v..v... Điều này dẫn đến hậu quả là sự mất thông tin. Vì vậy, hệ quản trị cơ sở dữ liệu phải có các cơ chế đáp ứng lại nguy cơ hệ thống bị hỏng hóc, nhằm đảm bảo ***tính nguyên tử và tính lâu bền*** của các giao dịch. Chương này trình bày các nguyên lý của một hệ thống phục hồi nhằm khôi phục CSDL đến một trạng thái nhất quán trước khi xảy ra sự cố.

YÊU CẦU

Hiểu rõ các sự cố có thể xảy ra trong đời sống của một cơ sở dữ liệu, các nguyên nhân của sự không nhất quán dữ liệu.

Hiểu các kỹ thuật phục hồi, các ưu nhược điểm của mỗi kỹ thuật.

PHÂN LỚP HỎNG HÓC:

Có nhiều kiểu hỏng hóc có thể xảy đến với hệ thống, mỗi một trong chúng cần được ứng xử một cách riêng biệt. Trong chương này ta chỉ xét các kiểu hỏng hóc sau:

- **Hỏng hóc trong giao dịch**: Có hai loại lỗi làm cho giao dịch bị hỏng hóc: **1. Lỗi luận lý**: Giao dịch không thể tiếp tục thực hiện bình thường được nữa do một số điều kiện bên trong không được thoả. ví dụ như: dữ liệu đầu vào không đúng, không tìm thấy dữ liệu, trào dữ liệu hoặc do việc sử dụng tài nguyên vượt hạn định. **2. Lỗi hệ thống:** Hệ thống rơi vào trạng thái không mong muốn ví dụ như trạng thái deadlock.
- **Hệ thống bị hư hỏng:** Có một phần cứng sai chức năng hoặc có một sai sót trong phần mềm cơ sở dữ liệu hay hệ điều hành.
- **Đĩa bị hư hỏng:** Một khối đĩa bị mất nội dung.

Để hệ thống có thể đề ra được chiến lược phục hồi lỗi phù hợp, trước tiên cần phải xác định các loại hỏng hóc trên các thiết bị lưu trữ dữ liệu. Sau đó, cần xác định những hỏng hóc này ảnh hưởng như thế nào đến nội dung cơ sở dữ liệu. Nhiệm vụ quan trọng sau cùng là đề ra các giải pháp nhằm đảm bảo tính nhất quán của cơ sở dữ liệu và tính nguyên tử của giao dịch mỗi khi hỏng hóc đã phát sinh. Các giải pháp này thường được gọi là các giải thuật phục hồi (recovery algorithms).

Các giải thuật phục hồi gồm có hai phần:

1. Các hành động được thực hiện trong suốt quá trình hoạt động bình thường của giao dịch nhằm đảm bảo có đầy đủ thông tin cho việc phục hồi sau này.
2. Các hành động được thực hiện sau khi lỗi phát sinh. Nhằm khôi phục nội dung của cơ sở dữ liệu trở về một trạng thái trước đó, và trạng thái này thoã mãn được các yêu cầu về tính nhất quán của cơ sở dữ liệu, tính bền và tính nguyên tử của giao dịch .

CẤU TRÚC LƯU TRỮ:

Như đã xét trong chương II, các hạng mục dữ liệu khác nhau của cơ sở dữ liệu có thể được lưu trên nhiều phương tiện lưu trữ khác nhau. Để nắm được cách thức đảm bảo tính nguyên tử và tính lâu bền của một giao dịch, cần phải có cái nhìn sâu hơn về các loại thiết bị lưu trữ dữ liệu và cách thức truy xuất chúng.

CÁC LOẠI LƯU TRỮ:

- **Lưu trữ không ổn định (volatile storage):** Thông tin lưu trong thiết bị lưu trữ không ổn định sẽ bị mất khi hệ thống bị hỏng hóc. Ví dụ của thiết bị lưu trữ không ổn định là: bộ nhớ chính, bộ nhớ cache. Sự truy cập đến các thiết bị lưu trữ không ổn định là cực nhanh. Lý do: một là: do tính chất của bộ nhớ cho phép như vậy; hai là: có thể truy xuất trực tiếp các hạng mục dữ liệu chứa trong nó.
- **Lưu trữ ổn định (nonvolatile storage):** Thông tin lưu trữ trong thiết bị lưu trữ ổn định thường không bị mất khi hệ thống bị sự cố. Tuy nhiên, nguy cơ bản thân thiết bị lưu trữ ổn định bị hỏng vẫn có thể xảy ra. Ví dụ của thiết bị lưu trữ ổn định là: đĩa từ và băng từ. Trong hầu hết các hệ cơ sở dữ liệu, thiết bị lưu trữ ổn định thường được dùng là đĩa từ. Các loại thiết bị lưu trữ ổn định khác được dùng để lưu trữ phòng hờ (back up) dữ liệu.

- **Lưu trữ bền (stable storage):** Theo lý thuyết thì thông tin chứa trong thiết bị lưu trữ bền ***không bao giờ*** bị mất khi hệ thống bị hư hỏng. Tuy nhiên, trong thực tế, ta khó lòng tạo ra được một thiết bị đạt được tính chất lý tưởng như vậy. Chỉ có giải pháp tăng cường độ bền mà thôi.

THỰC THI LƯU TRỮ BỀN:

Tiêu chí để thực hiện việc lưu trữ bền là nhân bản thông tin cần thiết trong một vài phương tiện lưu trữ ổn định khác nhau với các phương thức hỏng hóc độc lập và cập nhật các phiên bản thông tin này một cách có tổ chức, sao cho dù có lỗi xuất hiện trong quá trình chuyển dữ liệu, thông tin vẫn không bị hư hại.

- Các hệ thống RAID đảm bảo rằng việc hỏng hóc của một đĩa không gây sự mất dữ liệu. Dạng thức đơn giản và nhanh nhất của RAID là dùng đĩa gương (mirrored disk). Các dạng thức khác giúp tiết kiệm chi phí, nhưng cái giá phải trả là thời gian đọc ghi chậm hơn.
- Tuy nhiên các hệ thống RAID vẫn không đảm bảo được tính an toàn dữ liệu khi gặp phải tai họa như: cháy nổ, lụt lội. Người ta đề nghị một hệ thống lưu trữ mới an toàn hơn hoạt động theo nguyên tắc sau: Sao lưu dữ liệu sang một vài vị trí địa lý khác nhau thông qua mạng máy tính.

Sau đây là cách thức đảm bảo thông tin lưu trữ không bị lỗi trong quá trình đọc ghi dữ liệu:

Việc chuyển một khối dữ liệu giữa bộ nhớ và đĩa có thể dẫn đến kết quả:

- ***Thành công hoàn toàn***: Thông tin được chuyển đến đích an toàn.
- ***Bị lỗi một phần***: Có lỗi xuất hiện trong quá trình chuyển dữ liệu và khối đích chứa thông tin không đúng.
- ***Bị lỗi hoàn toàn***: Lỗi xuất hiện ngay ở giai đoạn đầu của quá trình truyền dữ liệu. Khối đích giữ nguyên như ban đầu.

Nếu có lỗi xuất hiện trong quá trình truyền dữ liệu, hệ thống phải phát hiện được và thực thi thủ tục phục hồi lỗi. Để làm được như vậy, hệ thống phải duy trì hai khối dữ liệu vật lý cho mỗ khối dữ liệu luận lý. (Trong tình huống dùng hệ thống đĩa gương thì hai khối vật lý này ở cùng một địa điểm, trong tình huống dùng hệ thống sao lưu từ xa, hai khối này ở hai địa điểm khác nhau).

Một thao tác ghi dữ liệu được thực thi như sau:

1. Viết thông tin lên khối vật lý thứ nhất.
2. Khi hành động ghi thứ nhất thành công, tiếp tục ghi phần thông tin trên lên khối vật lý thứ hai.
3. Thao tác ghi được coi là thành công khi thao tác ghi thứ hai thành công.

Trong quá trình phục hồi, từng cặp khối vật lý được kiểm tra:

1. Nếu nội dung của cả hai như nhau và không có lỗi có thể phát hiện, khi đó không cần làm gì thêm.
2. Nếu một trong hai khối có lỗi phát hiện được, khi đó thay thế khối bị lỗi bởi nội dung của khối còn lại.
3. Nếu cả hai khối không có lỗi phát hiện được, nhưng nội dung của chúng khác nhau, thay thế khối thứ nhất bởi khối thứ hai.

Yêu cầu phải so sánh từng cặp khối vật lý một đòi hỏi phải mất nhiều thời gian. Người ta có thể cải thiện tình huống này bằng cách lưu vết những thao tác viết khối trong tiến trình thực thi. Khi phục hồi, chỉ những khối nào thao tác ghi ở trong tiến trình thực thi mới cần được đem so sánh. Giao thức để viết ra một khối đến một site xa tương tự như viết khối trong hệ thống đĩa gương..

TRUY CẬP DỮ LIỆU

Như đã xét trong chương II, hệ cơ sở dữ liệu nằm thường trực trên các thiết bị lưu trữ ổn định (thường là đĩa từ) và thường được phân thành các đơn vị lưu trữ kích thước cố định được gọi là khối (blocks). Khối là đơn vị truyền nhận dữ liệu từ/ra đĩa. Một khối có thể chứa vài hạng mục dữ liệu. Ta giả thiết không có hạng mục dữ liệu nào trải ra trên nhiều hơn một khối.

Các giao dịch nhập (input) thông tin từ đĩa vào bộ nhớ chính và xuất (output) thông tin theo chiều ngược lại. Các thao tác nhập/xuất này được thực hiện theo đơn vị khối. Khối nằm trên đĩa được gọi là khối vật lý (physical block), khối được trữ tạm trong bộ nhớ chính được gọi là khối đệm (buffer block). Vùng bộ nhớ tạm chứa các khối dữ liệu được gọi là vùng đệm đĩa (disk buffer).

Việc di chuyển khối giữa đĩa và bộ nhớ được thực hiện thông qua hai thao tác:

1. **Input(B)** chuyển khối vật lý B vào bộ nhớ chính.
2. **Output(B)** chuyển khối đệm B ra đĩa và thay thế cho khối vật lý tương ứng ở đó.

Hình dưới đây sẽ mô phỏng cho hai thao tác này

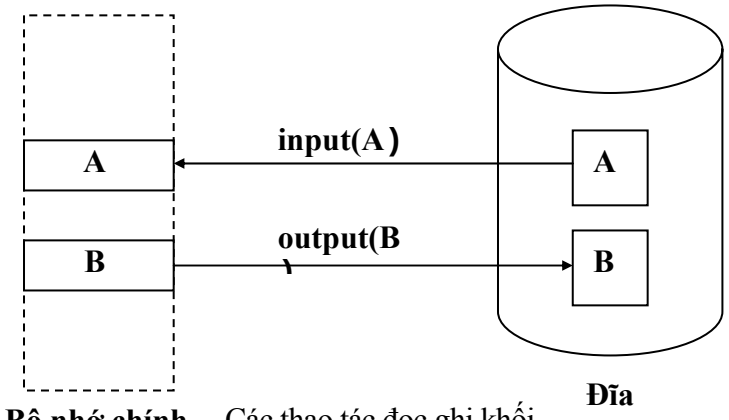

figure VI- 1

Mỗi giao dịch T_i có một vùng làm việc riêng ở đó các bản sao của tất cả các hạng mục dữ liệu được truy xuất và cập nhật được lưu giữ. Vùng làm việc này được tạo ra khi giao dịch khởi động. Nó bị xoá đi khi giao dịch bàn giao (commit) hoặc huỷ bỏ (abort). Mỗi hạng mục dữ liệu ***x*** được trữ trong vùng làm việc của giao dịch T_i sẽ được ký hiệu là x_i. Giao dịch T_i trao đổi với hệ cơ sở dữ liệu bằng cách chuyển dữ liệu đến/ra vùng làm việc của nó sang vùng đệm của hệ thống.

Hai thao tác dùng để chuyển dữ liệu:

1. **read(X)** gán giá trị của hạng mục dữ liệu X cho biến cục bộ x_i. Thao tác này được thực hiện như sau:
   a. Nếu khối B_X chứa X không có trong bộ nhớ chính thì thực hiện thao tác **input(B_X).**
   b. Gán cho x_i giá trị của X trong khối đệm.
2. **write(X)** gán giá trị của biến cục bộ x_i cho hạng mục dữ liệu X trong khối đệm. Thao tác này được thực hiện như sau:
   a. Nếu khối B_X chứa X không có trong bộ nhớ thì thực hiện thao tác **input(B_X).**
   b. Gán giá trị của x_i cho X trong vùng đệm B_X.

Chú ý rằng cả hai thao tác đều có thể đòi hỏi chuyển một khối từ đĩa vào bộ nhớ chính nhưng không yêu cầu chuyển một khối từ bộ nhớ chính ra đĩa.

Đôi khi một khối đệm bị ghi bắt buộc ra đĩa do bộ quản lý vùng đệm cần không gian bộ nhớ cho các mục đích khác hoặc do hệ cơ sở dữ liệu muốn phản ánh những thay đổi trong khối dữ

liệu B trên đĩa. Khi hệ cơ sở dữ liệu thực hiện thao tác **Output(B)** ta nói nó đã xuất bắt buộc khối đệm B ra đĩa.

Khi một giao dịch cần truy xuất hạng mục dữ liệu X lần đầu, nó phải thực hiện **Read(X)**. Khi đó tất cả các cập nhật đối với X được thực hiện trên x_i. Sau khi giao dịch truy xuất X lần cuối, nó thực hiện **Write(X)** để ghi lại sự thay đổi của X trong CSDL.

Không nhất thiết phải thực hiện thao tác **Output(B_X)** ngay sau khi thao tác **write(X)** hoàn thành. Lý do là: khối đệm B_X có thể còn chứa các hạng mục dữ liệu khác đang được truy xuất. Nếu hệ thống bị hư hỏng ngay sau khi thao tác **write(X)** hoàn thành, nhưng trước khi thực hiện thao tác **Output(B_X)**, giá trị mới của X sẽ không bao giờ được ghi ra đĩa, do đó, nó bị mất!

PHỤC HỒI VÀ TÍNH NGUYÊN TỬ:

Trở lại với ví dụ đơn giản về hệ thống ngân hàng: Giao dịch T_i thực hiện việc chuyển $50 từ tài khoản A sang tài khoản B. Giả sử giá trị ban đầu của các tài khoản A và B là $1000 và $2000. Giả sử hệ thống bị hư hỏng trong khi T_i đang thực thi: sau khi thao tác **output(B_A)** được thực hiện và trước khi thực hiện thao tác **output(B_B)** (B_A và B_B là hai khối đệm chứa hai hạng mục A và B). Người ta có thể thực hiện một trong hai giải pháp phục hồi sau:

1. Thực hiện lại T_i. Thủ tục này sẽ dẫn đến kết quả: giá trị của A là $900 thay vì phải là $950. Do đó, hệ thống ở trong trạng thái không nhất quán.
2. Không thực hiện lại T_i . Kết quả: giá trị của A và B tương ứng sẽ là $950 và $2000. Hệ thống cũng trong trạng thái không nhất quán.

Vấn đề phát sinh ở chỗ: T_i thực hiện nhiều thao tác sửa đổi nội dung cơ sở dữ liệu, do đó cần nhiều thao tác xuất dữ liệu ra đĩa, nhưng lỗi phát sinh không cho phép tất cả các thao tác xuất dữ liệu hoàn thành.

Giải pháp nhằm đạt được tính nguyên tử là: trước khi thực hiện các thao tác sửa đổi cơ sở dữ liệu, cần ghi ra các thiết bị lưu trữ bền những thông tin mô tả các sửa đổi này. Cụ thể của giải pháp trên sẽ được trình bày trong các phần V.4, V.5 và V.6.

PHỤC HỒI DỰA TRÊN SỔ GHI LỘ TRÌNH (Log-based recovery)

Một cấu trúc thường được dùng để ghi lại những thay đổi trên cơ sở dữ liệu là sổ ghi lộ trình (log). Log là một dãy các mẩu tin lộ trình (log records). Một thao tác cập nhật trên cơ sở dữ liệu sẽ được ghi nhận bằng một log record. Một log record kiểu mẫu chứa các trường sau:

- **Định danh giao dịch (transaction identifier)**: định danh duy nhất của giao dịch thực hiện hoạt động **write**
- **Định danh hạng mục dữ liệu (Data-item identifier)**: định danh duy nhất của hạng mục dữ liệu được viết (thường là vị trí của hạng mục dữ liệu trên đĩa)
- **Giá trị cũ (Old value)**: giá trị của hạng mục dữ liệu trước khi viết
- **Giá trị mới (New value)**: giá trị hạng mục dữ liệu sẽ có sau khi viết

Có một vài log record đặc biệt mang các ý nghĩa riêng. Bảng sau đây chỉ ra một số loại log record và ý nghĩa của chúng:

LOẠI LOG RECORD	Ý NGHĨA
< T_i **start** >	Giao dịch T_i đã khởi động.
< T_i, X_j, V_1, V_2 >	Giao dịch T_i đã thực hiện thao tác ghi trên hạng mục dữ liệu X_j, X_j có giá trị V_1 trước khi ghi và nhận giá trị V_2 sau khi ghi.
< T_i **commit** >	Giao dịch T_i đã bàn giao.
< T_i **abort** >	Giao dịch Ti đã huỷ bỏ.

Mỗi khi một giao dịch thực hiện một thao tác ghi, trước tiên phải tạo ra một log record cho thao tác ghi đó (trong log file), trước khi giao dịch thay đổi cơ sở dữ liệu. Như vậy, hệ thống có cơ sở để huỷ bỏ (undo) một thay đổi đã được làm trên cơ sở dữ liệu bằng cách sử dụng trường **Old-value** trong log record.

log phải được lưu trong những thiết bị lưu trữ bền. Mỗi một log record mới ngầm định sẽ được thêm vào cuối tập tin log.

CẬP NHẬT TRÌ HOÃN CƠ SỞ DỮ LIỆU (Deferred Database Modification):

Kỹ thuật cập nhật trì hoãn đảm bảo tính nguyên tử của giao dịch bằng cách ghi lại tất cả những sửa đổi cơ sở dữ liệu vào sổ ghi lộ trình (log), nhưng trì hoãn sự thực hiện tất cả các thao tác viết dữ liệu ra đĩa của giao dịch cho đến khi giao dịch bàn giao một phần (partially commits). Nhắc lại rằng: một giao dịch được gọi là bàn giao một phần khi hành động cuối cùng của nó được thực hiện xong. Kỹ thuật cập nhật trì hoãn được trình bày trong phần này giả thiết rằng các giao dịch được thực hiện một cách tuần tự.

Khi giao dịch bàn giao một phần, thông tin trên log kết hợp với giao dịch được sử dụng trong việc viết trì hoãn. Nếu hệ thống có sự cố trước khi giao dịch hoàn thành việc thực hiện của nó hoặc giao dịch bị bỏ dở khi đó thông tin trên log bị bỏ lơ.

Sự thực thi của một giao dịch được tiến triển như sau:

- Trước khi giao dịch T_i bắt đầu thực hiện, một mẫu tin < T_i **start** > được ghi ra sổ lộ trình.
- Trước khi T_i thực hiện thao tác **write(X),** một mẫu tin < T_i, X, V_2 > được ghi ra sổ lộ trình.
- Cuối cùng, khi giao dịch T_i bàn giao một phần, mẫu tin < T_i **commit** > được ghi ra sổ lộ trình.

Khi giao dịch bàn giao một phần, các mẫu tin trong sổ lộ trình kết hợp với giao dịch sẽ được sử dụng để thực hiện việc ghi trì hoãn các hạng mục dữ liệu ra đĩa. Nhà thiết kế phải đảm bảo rằng, trước khi hoạt động ghi hạng mục dữ liệu diễn ra, các mẫu tin log đã được ghi thành công ra các thiết bị lưu trữ bền. Ngoài ra cũng cần để ý: kỹ thuật cập nhật trì hoãn chỉ cần ghi lại giá trị mới của hạng mục dữ liệu (V_2) mà thôi.

Để minh hoạ, ta sử dụng ví dụ hệ thống ngân hàng đơn giản. Gọi T_0 là giao dịch có nhiệm vụ chuyển $50 từ tài khoản A sang tài khoản B, T_1 là giao dịch có nhiệm vụ rút $100 từ tài khoản C. Giả sử giá trị ban đầu của các tài khoản A, B, C là $1000, $2000 và $700. Hành động của T_0 và T_1 được mô tả như sau:

T_0	T_1
read(A)	Read(C)
A:=A-50	C:=C-100
write(A)	write(C)
read(B)	
B:=B+50	
write(B)	

Giả thiết các giao dịch được thực hiện tuần tự: T_0 rồi tới T_1. Một phần của sổ lộ trình ghi lại những thông tin liên quan đến hoạt động của hai giao dịch trên được cho trong bảng dưới đây:

<T_0 start>
<T_0 ,A, 950>
<T_0 ,B, 2050>

<T_0 commit>
<T_1 start>
<T_1 ,C, 600>
<T_1 commit>

figure VI- 2

Sau khi có sự cố xảy ra, hệ thống phục hồi sẽ tham khảo sổ lộ trình để chọn ra những giao dịch nào cần được làm lại (redo). Giao dịch T_i cần được làm lại khi và chỉ khi sổ nhật ký có chứa cả hai mẫu tin <T_i start> và <T_i commit>.

Thủ tục làm lại giao dịch T_i như sau:

- **redo(T_i)** đặt giá trị mới cho tất cả các hạng mục dữ liệu được cập nhật bởi giao dịch T_i. Các giá trị mới sẽ được tìm thấy trong sổ lộ trình (log).

Hoạt động redo phải đồng hiệu lực (idempotent) có nghĩa là việc thực hiện nó nhiều lần tương đương với việc thực hiện nó một lần.

Trở lại ví dụ vừa nêu, ta có bảng mô tả trạng thái của sổ ghi lộ trình và cơ sở dữ liệu như sau:

LOG	CƠ SỞ DỮ LIỆU
<T_0 start>	
<T_0 ,A, 950>	
< T_0 ,B, 2050>	
<T_0 commit>	
	A=950
	B=2050
<T_1 start>	
<T_1 ,C, 600>	
<T_1 commit>	
	C=600

figure VI- 3

Sau đây là một số tình huống mô phỏng:

1. Giả sử lỗi hệ thống xảy ra sau khi mẫu tin log cho hành động **write(B)** của giao dịch T_0 vừa được ghi ra thiết bị lưu trữ bền. Khi hệ thống khởi động trở lại, sẽ không có hành động "thực hiện lại giao dịch" nào cần phải làm, do không có mẫu tin ghi **commit** nào xuất hiện trong sổ lộ trình. Nghĩa là giá trị của A,B và C vẫn giữ nguyên là $1000, $2000 và $700.
2. Giả sử lỗi hệ thống xảy ra sau khi mẫu tin log cho hành động **write(C)** của giao dịch T_1 vừa được ghi ra thiết bị lưu trữ bền. Khi hệ thống hoạt động trở lại, thủ tục **redo(T_0)** sẽ được thực hiện do có sự xuất hiện của mẫu tin **<T_0 commit>** trong sổ lộ trình. Sau khi thủ tục này được thực thi, giá trị của A và B sẽ là $950 và $2050.

CẬP NHẬT TỨC THỜI CƠ SỞ DỮ LIỆU (Immediate Database Modification):

Kỹ thuật cập nhật tức thời cho phép các thao tác sửa đổi cơ sở dữ liệu có quyền xuất dữ liệu tức thời ra đĩa trong khi giao dịch vẫn còn ở trong trạng thái hoạt động (active state). Hành động thay đổi nội dung dữ liệu tức thời của các giao dịch đang hoạt động được gọi là "những thay đổi chưa được bàn giao" (uncommitted modifications).

Sự thực thi của một giao dịch được tiến hành như sau:

- Trước khi giao dịch T_i bắt đầu sự thực hiện, một mẫu tin < T_i **start** > được ghi ra sổ lộ trình.

- Trước khi T_i thực hiện thao tác **write(X),** một mẫu tin < T_i, X, V_1, V_2 > được ghi ra sổ lộ trình.
- Cuối cùng, khi giao dịch T_i bàn giao một phần, mẫu tin < T_i **commit** > được ghi ra sổ lộ trình.

Cần phải đảm bảo rằng, trước khi hoạt động ghi hạng mục dữ liệu diễn ra, các mẫu tin log đã được ghi thành công ra các thiết bị lưu trữ bền. Ngoài ra, cũng cần chú ý là mẫu tin log cho hành động **write(X)** của giao dịch T_i, tức là mẫu tin <T_i, X, V_1, V_2> có chứa cả hai giá trị mới (V_2) và cũ (V_1) của hạng mục dữ liệu X.

Trở lại với ví dụ trong phần V.4.1, ta có một phần của sổ lộ trình liên quan đến các hoạt động của T_0 và T_1 như sau:

<T_0 start>
<T_0 , A, 1000, 950>
<T_0 , B, 2000, 2050>
<T_0 commit>
<T_1 start>
<T_1 , C, 700, 600>
<T_1 commit>

figure VI- 4

Bảng mô tả trạng thái của sổ ghi lộ trình và cơ sở dữ liệu như sau:

LOG	CƠ SỞ DỮ LIỆU
<T_0 start>	
<T_0 , A, 1000, 950>	
< T_0 , B, 2000, 2050>	
	A=950
	B=2050
<T_0 commit>	
<T_1 start>	
<T_1 , C, 700, 600>	
	C=600
<T_1 commit>	

figure VI- 5

Kỹ thuật cập nhật tức thời sử dụng hai thủ tục khôi phục sau lỗi:

- **undo(T_i)** đặt lại giá trị cũ cho tất cả các hạng mục dữ liệu được cập nhật bởi giao dịch T_i. Các giá trị cũ sẽ được tìm thấy trong sổ lộ trình (log).
- **redo(T_i)** đặt giá trị mới cho tất cả các hạng mục dữ liệu được cập nhật bởi giao dịch T_i. Các giá trị mới sẽ được tìm thấy trong sổ lộ trình (log).

Sau khi lỗi xuất hiện, hệ thống phục hồi tham khảo sổ ghi để quyết định những giao dịch nào cần được làm lại (redo) và những giao dịch nào cần được huỷ bỏ (undo).

- Giao dịch T_i cần được huỷ bỏ khi sổ ghi chứa mẫu tin <T_i start> nhưng không có mẫu tin <T_i commit>.
- Giao dịch T_i cần được làm lại khi sổ ghi có chứa cả mẫu tin <T_i start> lẫn mẫu tin <T_i commit>.

Sau đây là một số tình huống mô phỏng:

1. Giả sử lỗi hệ thống xảy ra sau khi mẫu tin log cho hành động **write(B)** của giao dịch T_0 vừa được ghi ra thiết bị lưu trữ bền. Khi hệ thống khởi động trở lại, nó sẽ tìm thấy mẫu tin <T_0 start> trong sổ ghi, nhưng không có mẫu tin <T_0 commit> tương ứng. Do đó giao dịch T_0 cần phải được huỷ bỏ. Nghĩa là thủ tục **undo(T_0)** sẽ được gọi và giá trị của A,B và C vẫn giữ nguyên là \$1000, \$2000 và \$700.
2. Giả sử lỗi hệ thống xảy ra sau khi mẫu tin log cho hành động **write(C)** của giao dịch T_1 vừa được ghi ra thiết bị lưu trữ bền. Khi hệ thống hoạt động trở lại, hai thủ tục **redo(T_0)** và **undo(T_1)** sẽ được thực hiện. Do có sự xuất hiện của các mẫu tin **<T_0 start>**, **<T_0 commit>**, **<T_1 start>** trong sổ lộ trình. Sau khi hai thủ tục này được thực thi, giá trị của A, B, C sẽ là \$950,ì \$2050 và \$700.

ĐIỂM KIỂM SOÁT (Checkpoint):

Khi lỗi hệ thống xuất hiện, hệ thống phục hồi phải tham khảo sổ ghi lộ trình để quyết định những giao dịch nào cần được làm lại và những giao dịch nào cần được huỷ bỏ. Theo nguyên lý thì cần phải tìm kiếm toàn bộ nội dung của sổ ghi để có được quyết định trên.

Hướng tiếp cận trên sẽ gặp phải hai khó khăn lớn:

1. Quá trình tìm kiếm mất nhiều thời gian.
2. Theo các giải thuật vừa nêu, hầu hết các giao dịch cần được làm lại đã ghi những dữ liệu được cập nhật ra cơ sở dữ liệu rồi. Việc làm lại chúng tuy không có hại gì, nhưng lại làm cho tiến trình khôi phục trở nên lâu hơn.

Công cụ "điểm kiểm soát" (checkpoint) được sử dụng để cải thiện hiệu năng của quá trình khôi phục. Trong quá trình hoạt động của mình, hệ thống sẽ duy trì một sổ ghi lộ trình bằng cách sử dụng một trong hai kỹ thuật được giới thiệu trong phần V.4.1 và V.4.2. Ngoài ra, hệ thống còn phải thực hiện một cách chu kỳ các hành động đặt điểm kiểm soát. Hành động này đòi hỏi một dãy các thao tác sau:

1. Xuất ra lưu trữ bền tất cả các mẫu tin ghi nhận lộ trình (log record) đang nằm trong bộ nhớ chính.
2. Xuất ra đĩa tất cả những khối đệm đã được cập nhật.
3. Xuất ra thiết bị lưu trữ bền một log-record <**checkpoint**>

Các giao dịch sẽ không được phép thực hiện bất kỳ thao tác cập nhật dữ liệu nào (ví dụ như ghi các khối đệm, ghi các mẫu tin log) khi hành động đặt điểm kiểm soát đang được thực hiện.

Sự hiện diện của điểm kiểm soát trong sổ ghi cho phép hệ thống tổ chức quá trình phục hồi tốt hơn. Xét một giao dịch T_i đã bàn giao (commit) trước một điểm kiểm soát. Ta có mẫu tin **<T_i commit>** xuất hiện trước mẫu tin **<checkpoint>.** Có nghĩa là tất cả các thay đổi mà T_i đã làm đối với cơ sở dữ liệu phải được thực hiện trước khi người ta đặt điểm kiểm soát trên. Vì vậy, trong giai đoạn phục hồi sau lỗi, người ta không cần phải làm lại (**redo)** giao dịch T_i.

Dựa trên điểm cải tiến này, ta cải tiến lại các kỹ thuật đã được trình bày trong phần V.4.1 và V.4.2 như sau:

1. Sau khi lỗi hệ thống xuất hiện, hệ thống phục hồi sẽ kiểm tra lại sổ lộ trình (log) để tìm ra giao dịch T_i thoả điều kiện: đó là giao dịch gần đây nhất được khởi động trước điểm kiểm soát gần đây nhất. Qui trình tìm T_i như sau: dò ngược trong sổ ghi lộ trình cho đến khi tìm thấy mẫu tin <**checkpoint**> đầu tiên. Từ điểm kiểm soát này, lại tiếp tục dò ngược trong sổ ghi cho đến khi tìm thấy mẫu tin <**T_i start**> đầu tiên. Mẫu tin này chỉ ra giao dịch T_i .
2. Khi đã xác định được giao dịch T_i rồi, các thủ tục **undo** và **redo** chỉ được áp dụng cho giao dịch T_i và các giao dịch diễn ra sau T_i. Chúng ta ký hiệu tập những giao dịch vừa nói là **T**.
3. Với kỹ thuật "Cập nhật tức thời cơ sở dữ liệu", tiến trình phục hồi như sau:

- Với mọi giao dịch $T_k \in \mathbf{T}$ mà không có mẫu tin <**T_k commit**> trong sổ ghi lộ trình, thực thi **undo(T_k)**.
- Với mọi giao dịch $T_k \in T$ mà có mẫu tin <**T_k commit**> trong sổ ghi lộ trình, thực thi **redo(T_k)**.

4. Không cần thực thi thao tác **undo** khi sử dụng kỹ thuật "Cập nhật có trì hoãn cơ sở dữ liệu".

PHÂN TRANG BÓNG (Shadow Paging):

Kỹ thuật "Phân trang bóng" cũng là kỹ thuật cho phép phục hồi sau lỗi, nhưng ý tưởng thực hiện khác với các kỹ thuật dựa trên sổ ghi lộ trình vừa trình bày ở phần trên.
Sau đây là một số khái niệm cần được giải trình:

- ***Trang (page) là gì***? Như đã trình bày ở các phần trước, cơ sở dữ liệu được lưu vào thiết bị lưu trữ không phai thành nhiều khối có kích thước cố định. Người ta gọi những khối này là trang (page).
- ***Bảng trang và ý nghĩa của nó***: Khái niệm trang đã nói được mượn từ lý thuyết về Hệ điều hành. Cách quản lý trang cũng được thừa kế từ đó. Giả sử rằng cơ sở dữ liệu được phân thành n trang và sự phân bố trên đĩa của chúng có thể không theo một thứ tự cụ thể nào cả. Tuy nhiên, phải có cách để tìm ra nhanh và đúng trang thứ i của cơ sở dữ liệu ($1 \le i \le n$). Người ta dùng bảng trang (được mô phỏng như trong hình 5.2) cho mục đích này. Bảng trang có n đầu vào (entry). Mỗi đầu vào ứng với một trang. Một đầu vào chứa một con trỏ, trỏ đến một trang trên đĩa. Đầu vào đầu tiên chỉ đến trang đầu tiên của cơ sở dữ liệu, đầu vào thứ hai chỉ đến trang thứ hai ...

Ý tưởng then chốt của kỹ thuật "Phân trang bóng" là người ta sẽ duy trì hai bảng trang trong suốt chu kỳ sống của giao dịch, một bảng trang gọi là "*bảng trang hiện hành"* (current page table), bảng trang còn lại gọi là "*bảng trang bóng"* (shadow page table). Khi giao dịch khởi động, hai bảng trang này giống nhau. Bảng trang bóng sẽ không thay đổi suốt quá trình hoạt động của giao dịch. Bảng trang hiện hành sẽ bị thay đổi mỗi khi giao dịch thực hiện tác vụ **write.** Tất cả các tác vụ **input** và **output** đều sử dụng bảng trang hiện hành để định vị các trang trong đĩa. Điểm quan trọng khác là nên lưu bảng trang bóng vào thiết bị lưu trữ bền.

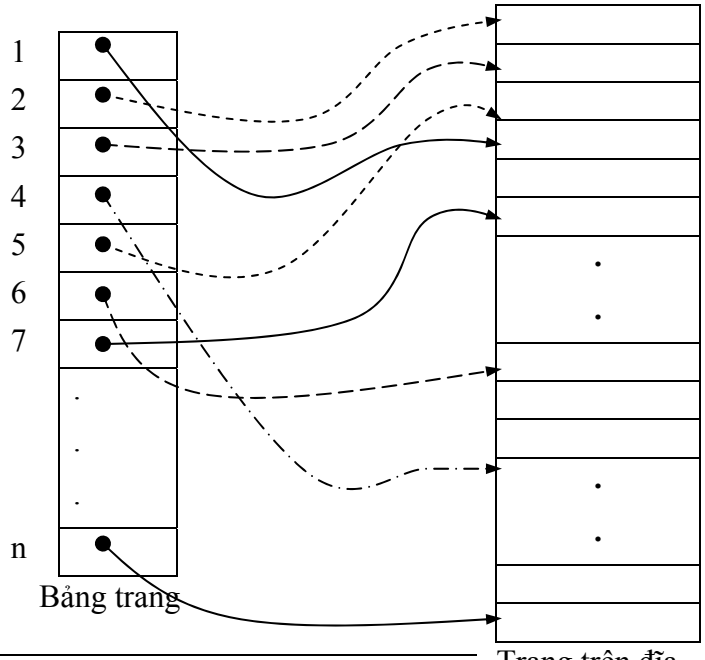

figure VI- 6

Giả sử giao dịch thực hiện tác vụ **write(X)** và hạng mục dữ liệu X được chứa trong trang thứ ***i.*** Tác vụ **write** được thực thi như sau:

1. Nếu trang thứ ***i*** chưa có trong bộ nhớ chính, thực hiện **input(X).**
2. Nếu đây là lệnh ghi được thực hiện ***lần đầu tiên*** trên trang thứ ***i*** bởi giao dịch, sửa đổi bảng trang hiện hành như sau:
   a. Tìm một trang chưa được dùng trên đĩa.
   b. Xoá trang vừa được tìm xong ở bước 2.a khỏi danh sách các khung trang tự do.
   c. Sửa lại bảng trang hiện hành sao cho đầu vào thứ ***i*** trỏ đến trang mới vừa tìm được trong bước 2.a.
3. Gán giá trị x_i cho X trong trang đệm (buffer page).

Để bàn giao một giao dịch, cần làm các bước sau:

1. Đảm bảo rằng tất cả các trang đệm trong bộ nhớ chính đã được giao dịch sửa đổi phải được xuất ra đĩa.
2. Xuất bảng trang hiện hành ra đĩa. chú ý là không được viết đè lên trang bóng
3. Xuất địa chỉ đĩa của bảng trang hiện hành ra vị trí cố định trong thiết bị lưu trữ bền. Vị trí này chính là nơi chứa địa chỉ của bảng trang bóng. Hành động này sẽ ghi đè lên địa chỉ của bảng trang bóng cũ. Như vậy, bảng trang hiện hành sẽ trở thành bảng trang bóng và giao dịch được bàn giao.

Nếu sự cố xảy ra trước khi hoàn thành bước thứ 3, hệ thống sẽ trở về trạng thái trước khi giao dịch được thực hiện. Nếu sự cố xảy ra sau khi bước thứ 3 hoàn thành, hiệu quả của giao dịch được bảo tồn; không cần thực hiện thao tác **redo** nào cả. Ví dụ trong hình 5.3 dưới đây mô phỏng lại trạng thái của các bảng trang hiện hành và bảng trang bóng khi giao dịch thực hiện thao tác ghi lên trang thứ tư của cơ sở dữ liệu có 10 trang.

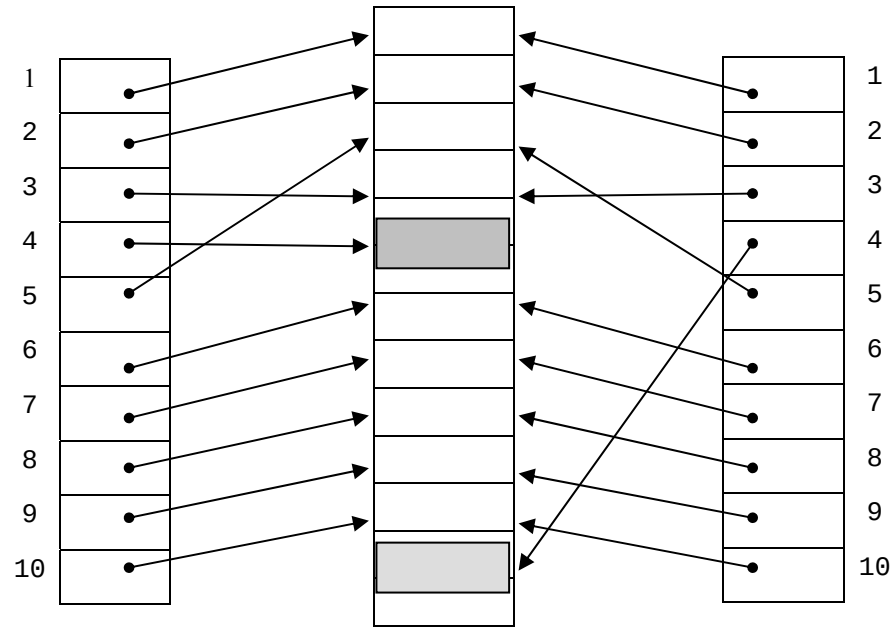

figure VI- 7 Ví dụ về bảng trang bóng và bảng trang hiện hành

Kỹ thuật phân trang bóng có một số điểm lợi hơn so với các kỹ thuật dựa trên sổ ghi:

1. Không mất thời gian ghi ra các log record.
2. Khôi phục sau sự cố nhanh hơn, do không cần các thao tác **undo** hoặc **redo**.

Tuy nhiên kỹ thuật phân trang bóng lại có nhiều nhược điểm:

- **Tổng phí bàn giao.** Xuất nhiều khối ra đĩa: các khối dữ liệu hiện tại, bảng trang hiện hành, địa chỉ của bảng trang hiện hành. Trong kỹ thuật dựa vào sổ ghi, chỉ cần xuất ra các log record, mà thông thường, các log record này vừa đủ chứa trong một khối.
- **Sự phân mảnh dữ liệu**. Trong chương II có trình bày chiến lược gom cụm vật lý các trang dữ liệu có liên quan với nhau. Sự gom cụm này cho phép việc vận chuyển dữ liệu nhanh hơn. Kỹ thuật phân trang bóng lại đổi vị trí của trang khi trang này bị sửa đổi. Điều này dẫn đến tính gom cụm dữ liệu không còn, hoặc phải dùng các giải pháp gom cụm lại rất mất thời gian.
- **Phải thu nhặt rác**. Mỗi khi giao dịch bàn giao, các trang chứa giá trị dữ liệu cũ đã bị sửa đổi bởi giao dịch sẽ trở thành không truy xuất được. Vì chúng không thuộc danh sách các trang tự do nhưng cũng không chứa dữ liệu hữu dụng. Ta gọi chúng là "rác". Cần thiết phải định kỳ tìm kiếm và thêm các trang rác vào trong danh sách các trang tự do. Hành động này được gọi là "thu nhặt rác".
- Ngoài ra, kỹ thuật phân trang bóng sẽ gặp nhiều khó khăn hơn kỹ thuật dựa vào sổ ghi khi cần được tinh chỉnh để đáp ứng cho yêu cầu phục vụ song song cho nhiều giao dịch. Vì những lý do trên, kỹ thuật phân trang bóng không được sử dụng rộng rãi lắm.

PHỤC HỒI VỚI CÁC GIAO DỊCH CẠNH TRANH

Cho dến bây giờ, ta chỉ xét các kỹ thuật phục hồi áp dụng cho các giao dịch được thực thi tuần tự. Bây giờ chúng ta sẽ tìm cách cải tiến kỹ thuật dựa vào sổ ghi nhằm đáp ứng yêu cầu phục vụ đồng thời cho nhiều giao dịch cạnh tranh. Ý tưởng thực hiện là: Không quan tâm đến số lượng các giao dịch cạnh tranh, hệ thống vẫn sử dụng một vùng đệm đĩa và một sổ ghi lộ trình. Các khối đệm được chia sẻ bởi tất cả các giao dịch. Chúng ta sẽ cho phép việc cập nhật tức thời cơ sở dữ liệu và cho phép một khối đệm có nhiều hạng mục dữ liệu được cập nhật bởi một hoặc nhiều giao dịch.

TRAO ĐỔI VỚI ĐIỀU KHIỂN CẠNH TRANH

Cơ chế phục hồi phụ thuộc rất nhiều vào cơ chế điều khiển cạnh tranh được sử dụng. Để cuộn lại một giao dịch thất bại (failed transaction), người ta phải huỷ bỏ (undo) các cập nhật được thực hiện bởi giao dịch. Giả sử giao dịch T_0 phải bị cuộn lại và một hạng mục dữ liệu Q đã bị T_0 thay đổi giá trị và cần phải được đặt lại giá trị cũ. Bằng cách sử dụng kỹ thuật dựa vào sổ ghi lộ trình, ta trả lại giá trị cũ cho Q bằng cách sử dụng một log record. Giả thiết lại có giao dịch thứ hai T_1 cũng vừa cập nhật Q xong, trước khi T_0 bị cuộn lại. Như vậy, sự cập nhật được thực hiện bởi T_1 sẽ bị mất đi nếu T_0 bị cuộn lại.

Biện pháp khắc phục là: nếu một giao dịch T đã cập nhật một hạng mục dữ liệu Q, thì không một giao dịch nào khác có quyền cập nhật lên hạng mục dữ liệu đó trong khi T chưa bàn giao hoặc chưa bị cuộn lại. Chúng ta có thể đảm bảo yêu cầu trên được thoả bằng cách sử dụng kỹ thuật "chốt hai kỳ nghiêm ngặt" (strict two-phase locking).

CUỘN LẠI GIAO DỊCH:

Phương pháp để cuộn lại (rollback) một giao dịch T_i, sử dụng sổ ghi, trong môi trường cạnh tranh như sau:

1. Dò ngược sổ ghi lộ trình để tìm ra các log record có dạng <T_i, Xj, V_1, V_2>.

2. Hạng mục dữ liệu Xj sẽ được trả lại giá trị cũ V_1.
3. Việc dò tìm kết thúc khi tìm thấy mẫu tin <T_i start>.

Việc dò ngược sổ ghi lộ ¹eó một ý nghĩa rất quan trọng, do một giao dịch có thể đã cập nhật một hạng mục dữ liệu nhiều hơn một lần. Một ví dụ: Xét một cặp log records như sau:

<T_i, A, 10, 20>
<T_i, A, 20, 30>

Cặp mẫu tin này thể hiện hai hành động cập nhật hạng mục dữ liệu A của giao dịch T_i. Nếu dò ngược sổ ghi lộ trình, A sẽ được trả về giá trị đúng là 10. Ngược lại, A sẽ nhận giá trị sai là 20.

Nếu kỹ thuật strict two-phase locking được sử dụng để điều khiển cạnh tranh, thì việc trả về giá trị cũ cho một hạng mục dữ liệu sẽ không xoá đi những tác động của các giao dịch khác lên hạng mục dữ liệu này.

CÁC ĐIỂM KIỂM SOÁT

Ở phần V.4.3, người ta đã sử dụng điểm kiểm soát (checkpoint) để làm giảm số lượng các log record mà hệ thống phục hồi phải dò tìm trong sổ ghi trong giai đoạn phục hồi sau lỗi. Nhưng, do đã giả thiết là không có cạnh tranh nên giải pháp V.4.3 chỉ xét đến những giao dịch sau trong quá trình khôi phục lỗi:

- Những giao dịch được khởi động sau điểm kiểm soát gần đây nhất.
- Một giao dịch (nếu có) đang trong trạng thái hoạt động (active) tại thời điểm người ta đặt điểm kiểm soát gần đây nhất.

Tình huống càng phức tạp khi các giao dịch được thực thi cạnh tranh. Có nghĩa là tại thời điểm đặt điểm kiểm soát, có thể có nhiều giao dịch đang ở trong trạng thái hoạt động.

Trong một hệ thống xử lý các giao dịch cạnh tranh, ta yêu cầu rằng: một mẫu tin ghi dấu kiểm soát (checkpoint log record) phải có dạng như sau:

<**checkpoint L**>

Trong đó L là danh sách các giao dịch đang hoạt động tại thời điểm đặt điểm kiểm soát.

Một lần nữa, ta qui ước rằng: khi hành động đặt điểm kiểm soát đang diễn ra, các giao dịch không được phép thực hiện bất kỳ thao tác cập nhật dữ liệu nào cả trên các khối đệm lẫn trên sổ ghi lộ trình.

Tuy nhiên, qui ước trên lại gây phiền toái, bởi vì các giao dịch phải ngừng hoạt động khi đặt điểm kiểm soát. Một kỹ thuật nâng cao giải quyết điểm phiền toái này là "Điểm kiểm soát mờ" (fuzzy checkpoint).

PHỤC HỒI KHỞI ĐỘNG LẠI (Restart Recovery)

Khi hệ thống phục hồi sau lỗi, nó tạo ra hai danh sách: *undo-list* bao gồm các giao dịch cần phải huỷ bỏ và *redo-list* bao gồm danh sách các giao dịch cần được làm lại.

Qui trình tạo lập hai danh sách *redo-list, undo-list* được thực hiện như sau:

1. Đầu tiên, chúng sẽ rỗng.
2. Dò ngược sổ ghi lộ trình, kiểm tra các mẫu tin cho đến khi tìm được mẫu tin <**checkpoint**> đầu tiên:
   - **a.** Với mỗi mẫu tin được tìm thấy theo dạng <**T_i commit**>, ta thêm T_i vào trong *redo-list.*
   - **b.** Với mỗi mẫu tin được tìm thấy theo dạng <**T_i start**>, nếu T_i không thuộc *redo-list* thì thêm T_i vào trong *undo-list.*
   - **c.** Khi tất cả các log record đã được xem xét, ta kiểm tra danh sách **L** trong mẫu tin <**checkpoint L**>. Với mọi giao dịch T_i trong **L**, nếu T_i không thuộc *redo-list* thì thêm T_i vào *undo-list*.

Khi hai danh sách *redo-list, undo-list* được thiết lập xong, tiến trình phục hồi được tiến hành như sau:

1. Dò ngược lại sổ ghi và thực hiện thủ tục **undo** đối với mỗi log record thuộc về giao dịch T_i trong *undo-list.* Các log record của các giao dịch nằm trong danh sách *redo-list* sẽ bị bỏ qua trong giai đoạn này. Việc dò ngược sẽ ngưng khi mẫu tin <**T_i start**> được tìm thấy cho mọi giao dịch T_i thuộc danh sách *undo-list.*
2. Định vị mẫu tin <**checkpoint L**> gần đây nhất trong log.
3. Dò sổ ghi theo chiều xuôi bắt đầu từ mẫu tin <**checkpoint L**> gần đây nhất và thực hiện thủ tục **redo** đối với mỗi log record thuộc về giao dịch T_i nằm trong danh sách *redo-list*. Trong giai đoạn này, bỏ qua các log record của các giao dịch thuộc danh sách *undo-list*.

Việc xử lý ngược ở bước 1 là rất quan trọng, nhằm đảm bảo kết quả trả về của cơ sở dữ liệu là đúng.

Sau khi tất cả các giao dịch trong danh sách *undo-list* bị huỷ bỏ, tất cả các giao dịch trong danh sách *redo-list* sẽ được làm lại. Sau khi tiến trình phục hồi thành công, xử lý giao dịch được tiếp tục.

Việc thực hiện huỷ bỏ các giao dịch trong *undo-list* trước khi làm lại các giao dịch trong *redo-list* có ý nghĩa rất quan trọng. Nếu làm ngược lại, vấn đề sau sẽ phát sinh: Giả sử hạng mục dữ liệu A có giá trị khởi đầu là 10. Giao dịch T_i đổi A thành 20 sau đó T_i bị huỷ bỏ. Sau đó, một giao dịch khác T_j cập nhật A thành 30. Đến đây thì hệ thống bị lỗi ngừng hoạt động. Hiện trạng của sổ ghi tại thời điểm hệ thống bị lỗi như sau:

<T_i, A, 10, 20>
<T_j, A, 10, 30>
<T_j commit>

Nếu thực hiện redo trước, A sẽ được đặt giá trị 30. Sau đó thực hiện undo, A sẽ được đặt giá trị 10, mà giá trị này sai. Giá trị cuối cùng của A phải là 30.

ĐIỂM KIỂM SOÁT MỜ (fuzzy checkpoint):

Kỹ thuật fuzzy checkpoint cho phép các giao dịch được cập nhật dữ liệu trên các khối đệm khi checkpoint-record đã được viết xong nhưng trước thời điểm các khối đệm đã sửa đổi được ghi ra đĩa.

Ý tưởng thực hiện fuzzy checkpoint như sau: Thay vì phải dò ngược sổ ghi để tìm mẫu tin checkpoint, ta sẽ lưu vị trí của mẫu tin checkpoint cuối cùng trong sổ ghi vào một chỗ cố định trong đĩa gọi là ***last_checkpoint***. Tuy nhiên, thông tin này sẽ không được cập nhật khi một mẫu tin checkpoint được ghi ra đĩa. Thay vào đó, trước khi một mẫu tin checkpoint được ghi ra sổ ghi, ta tạo ra một danh sách các khối đệm bị sửa đổi. Thông tin ***last_checkpoint*** được cập nhật chỉ sau khi tất cả các khối đệm bị sửa đổi được ghi ra đĩa.

last_checkpoint chỉ được dùng cho mục đích ***undo***.

BÀI TẬP CHƯƠNG VI

VI.1. Trình bày các điểm khác nhau giữa 3 kiểu lưu trữ: lưu trữ không ổn định, lưu trữ ổn định và lưu trữ bền theo tiêu chuẩn đánh giá là *chi phí cài đặt*.

VI.2. Thực tế, lưu trữ bền không thể thực hiện được. Giải thích tại sao? Hệ cơ sở dữ liệu giải quyết vấn đề này như thế nào?

VI.3. So sánh các kỹ thuật cập nhật tức thời và cập nhật có trì hoãn trong sơ đồ phục hồi dựa vào sổ ghi lộ trình theo các tiêu chuẩn: *tính dễ cài đặt và tổng chi phí thực hiện*.

VI.4. Giả sử rằng kỹ thuật cập nhật tức thời được sử dụng trong hệ thống. Bằng ví dụ, hãy chỉ ra rằng: tình trạng không nhất quán dữ liệu sẽ xảy ra nếu các log record không được ghi ra thiết bị lưu trữ bền trước khi giao dịch bàn giao (commit).

VI.5. Giải thích mục đích của cơ chế điểm kiểm soát (checkpoint). Hành động đặt điểm kiểm soát nên được thực hiện theo chu kỳ bao lâu là hợp lý?

VI.6. Khi hệ thống phục hồi sau lỗi, nó xây dựng 2 danh sách: *undo-list* và *redo-list*. Giải thích tại sao các log record của các giao dịch trong danh sách *undo-list* phải được xử lý theo thứ tự ngược, trong khi những log record trong danh sách *redo-list* lại được xử lý theo chiều xuôi?

VI.7. So sánh sơ đồ phục hồi phân trang bóng và sơ đồ phục hồi bằng sử dụng sổ ghi lộ trình theo tiêu chuẩn: *tính dễ cài đặt và tổng chi phí thực hiện*.

VI.8. Giả sử một cơ sở dữ liệu có 10 khối đĩa liên tiếp (khối 1, 2, 3, ..., 10). Hãy thể hiện trạng thái của buffer và thứ tự vật lý có thể có của các khối sau các thao tác cập nhật sau, giả sử: kỹ thuật phân trang bóng được sử dụng, buffer trong bộ nhớ chỉ đủ chứa 3 khối, chiến lược quản lý buffer là LRU (Least Recently Used)

Đọc khối 3
Đọc khối 7
Đọc khối 5

Đọc khối 3

Đọc khối 1

Sửa đổi khối 1

Đọc khối 10

Sửa khối 5