CHƯƠNG II

CÁC VẤN ĐỀ VÀ GIẢI PHÁP CƠ BẢN TRONG CÁC HỆ PHÂN TÁN

NỘI DUNG

- Truyền thông
- Định danh
- Đồng bộ
- Tiến trình trong các hệ thống phân tán
- Quản trị giao dịch và điều khiển tương tranh
- Phục hồi và chịu lỗi
- Bảo mật
- Tính nhất quán và vấn đề nhân bản

QUẢN TRỊ GIAO DỊCH VÀ ĐIỀU KHIỂN TƯƠNG TRANH

NỘI DUNG

- Giao tác và vấn đề tương tranh
- Các giải pháp điều khiển tương tranh

VÍ DỤ VỀ GIAO DỊCH NGÂN HÀNG

- Thao tác của khách hàng:
  deposit(amount): Gửi tiền vào tài khoản
  withdraw(amount): Rút tiền từ tài khoản
  getBalance() : Kiểm tra số tiền còn trong tài khoản
  setBalance(amount): Thiết lập số tiền cho tài khoản
- Thao tác của ngân hàng:
  create(name) : Tạo tài khoản mới
  lookUp(name) : Tìm kiếm tài khoản bằng tên
  branchTotal() : Tổng số tiền của các tài khoản

GIAO TÁC TRÊN ĐỐI TƯỢNG

- Quan niệm cơ bản: Chuỗi các yêu cầu của máy khách được thực hiện như một đơn vị riêng.
- Các thuộc tính của giao tác:
  - Nguyên tử (Atomic): Đối với thế giới bên ngoài thì giao tác không thể chia nhỏ hơn được. Các lệnh trong giao tác đều được thực hiện hoặc không có lệnh nào được thực hiện.
  - Nhất quán (Consistent): Giao tác không vi phạm tính bất biến hệ thống.
  - Cách ly (Isolated): Các giao tác đồng thời thực hiện cùng một lúc không ảnh hưởng lẫn nhau.
  - Có thời hạn (Durable): Khi đã xác nhận thay đổi thì những thay đổi đó là vĩnh cửu (Bền vững).
- Biện pháp xử lý (tối đa tương tranh): Đưa các yêu cầu xử lý xen kẽ

VÍ DỤ GIAO TÁC CỦA KHÁCH HÀNG

- Giả sử ba tài khoản A, B, C tương ứng với các biến a, b, c
- Các giao tác:
  - A rút $100 và chuyển vào tài khoản của B

    a.withdraw(100);

    b.deposit(100);
  - C rút $200 và chuyển vào tài khoản của B

    c.withdraw(200);

    b.deposit(200);

CÁC TÌNH HUỐNG LỖI

- Không ghi được vào thiết bị lưu trữ vĩnh cửu (Bộ nhớ ngoài: ổ đĩa, băng từ…)
- Bộ xử lý bị lỗi
- Thông điệp bị chậm hoặc bị thất lạc

Lỗi là chuyện bình thường

Không phát hiện ra lỗi mới bất bình thường

Giả thiết phát hiện được lỗi!

HOẠT ĐỘNG TRONG GIAO TÁC

- openTransaction() -> trans; Mở giao tác

  Bắt đầu giao tác và phân phát định danh của giao tác, định danh này sẽ được dùng trong các hoạt động khác của giao tác
- closeTransaction(trans) -> (commit, abort); Đóng giao tác //COMMIT, ABORT/ROLLBACK

  Kết thúc giao tác, nếu giá trị trả về là **commit** nghĩa là giao tác đã hoàn thành và dữ liệu đã được cập nhật theo đúng yêu cầu, nếu giá trị trả về là **abort** thì mọi hoạt động trong giao tác coi như chưa hề thực hiện
- abortTransaction(trans); Hủy giao tác

  Hủy bỏ giao tác //ABORT/ROLLBACK

QUÁ TRÌNH SỐNG CỦA GIAO TÁC

Thành công	Hủy bởi tiến trình khách	Hủy bởi tiến trình chủ
Mở giao tác	Mở giao tác	Mở giao tác
Thao tác	Thao tác	Thao tác
Thao tác	Thao tác	Thao tác
⋮	⋮	Tiến trình máy chủ hủy bỏ giao tác → ⋮
Thao tác	Thao tác	Thao tác: LỖI
Đóng giao tác	Hủy giao tác	Thông báo cho tiến trình khách

BEGIN TRANSACTION

INSERT INTO….

SET @Id=@@IDENTITY

ROLLBACK

VẤN ĐỀ TƯƠNG TRANH THƯỜNG GẶP

- Mất mát khi cập nhật (Lost update)
- Kết quả không đồng nhất

MẤT MÁT KHI CẬP NHẬT

Tài khoản của A, B, C có giá trị lần lượt là $100, $200 và $300.
A và C cùng chuyển cho B số tiền bằng 10% giá trị tài khoản của B

A chuyển cho B	C chuyển cho B

balance = b.getBalance();
b.setBalance(balance*1.1);
a.withdraw(balance/10)
	balance = b.getBalance();
	b.setBalance(balance*1.1);
	c.withdraw(balance/10)

balance = b.getBalance(); $200

balance = b.getBalance(); $200

b.setBalance(balance*1.1);$220

b.setBalance(balance*1.1);$220

a.withdraw(balance/10) $80

c.withdraw(balance/10) $280

Số tiền trong tài khoản của B lẽ ra phải là $242

KẾT QUẢ KHÔNG ĐỒNG NHẤT

A chuyển $100 cho B	Kiểm tra số dư tài khoản
a.withdraw(100) b.deposit(100)	aBranch.branchTotal()

a.withdraw(100); $100

total = a.getBalance() $100

total = total+b.getBalance() $300

total = total+c.getBalance()

b.deposit(100) $300

⋮

BIỆN PHÁP TUẦN TỰ HÓA THỰC HIỆN

Tổ hợp lại thứ tự thực hiện các yêu cầu

A chuyển cho B	C chuyển cho B

balance = b.getBalance();
b.setBalance(balance*1.1);
a.withdraw(balance/10)
	balance = b.getBalance();
	b.setBalance(balance*1.1);
	c.withdraw(balance/10)

balance = b.getBalance(); $200

b.setBalance(balance*1.1);$220

balance = b.getBalance(); $220

b.setBalance(balance*1.1);$242

a.withdraw(balance/10) $80

c.withdraw(balance/10) $278

Số tiền trong tài khoản của B là $242

BIỆN PHÁP TUẦN TỰ HÓA THỰC HIỆN

A chuyển $100 cho B	Kiểm tra số dư tài khoản
a.withdraw(100) b.deposit(100)	aBranch.branchTotal()

a.withdraw(100); $100

b.deposit(100) $300

total = a.getBalance() $100

total = total+b.getBalance() $400

total = total+c.getBalance()

⋮

ĐẢM BẢO TUẦN TỰ

- Hai thao tác được gọi là xung đột nếu kết quả tổ hợp thực hiện của chúng phụ thuộc vào thứ tự thực hiện.
- Cần phải xác định các thao tác có thể xảy ra xung đột
- Tất cả các cặp thao tác xung đột của các giao tác cần phải được thực hiện theo một thứ tự để đảm bảo tính tuần tự

CÁC LUẬT XUNG ĐỘT CHO THAO TÁC ĐỌC VÀ GHI

T1	T2	Xung đột	Nguyên nhân
Đọc	Đọc	Không	Kết quả của các cặp thao tác đọc không phụ thuộc vào thứ tự thực hiện của chúng
Đọc	Ghi	Có	Kết quả của thao tác đọc và ghi phụ thuộc vào thứ tự thực hiện của chúng
Ghi	Ghi	Có	Kết quả của các cặp thao tác ghi phụ thuộc vào thứ tự thực hiện của chúng

XEN KẼ THỰC HIỆN TƯƠNG ĐƯƠNG KHÔNG TUẦN TỰ

- Giả sử hai giao tác T và U thực hiện các thao tác sau:
  T: x = read(i); write(i, 10); write(j, 20)
  U: y = read(j); write(j, 30); z = read (i);
- Xen kẽ thực hiện tương đương không tuần tự như sau:

Giao tác T:	Giao tác U:
x = read(i)	
write(i, 10)	
	y = read(j)
	write(j, 30)
write(j, 20)	
	z = read (i)

KHẢ NĂNG PHỤC HỒI KHI HỦY THỰC HIỆN

- Máy chủ phải ghi nhận kết quả thực hiện của tất cả các thao tác trong giao tác khi COMMIT và hủy bỏ tất cả kết quả khi ABORT.
- Các vấn đề phục hồi bao gồm:
  - Đọc giá trị bẩn (Dirty reads)
  - Hủy phân tầng
  - Ghi trước
  - Thực hiện giao tác chặt chẽ
  - Sử dụng phiên bản tạm

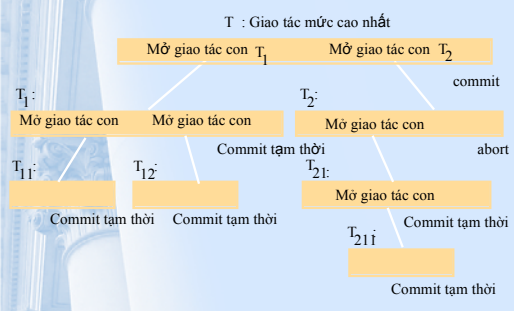
ĐỌC GIÁ TRỊ BẨN (Dirty Read)

Giao tác T		Giao tác U	
a.getBalance()		a.getBalance()	
a.setBalance(balance + 10)		a.setBalance(balance + 20)	
balance = a.getBalance()	$100		
a.setBalance(balance + 10)	$110		
		balance = a.getBalance()	$110
		a.setBalance(balance + 20)	$130
		commit transaction	
abort transaction			

GHI ĐÈ GIÁ TRỊ CHƯA COMMIT

Giao tác T		Giao tác U	
a.setBalance(105)		a.setBalance(110)	
	$100		
a.setBalance(105)	$105		
		a.setBalance(110)	$110

GIAO TÁC LỒNG NHAU



ƯU ĐIỂM CỦA GIAO TÁC LỒNG NHAU

- Các giao tác con trên cùng một mức có thể chạy đồng thời với nhau.
- Việc COMMIT hoặc ABORT của các giao tác con hoàn toàn độc lập với nhau.

CÁC PHƯƠNG PHÁP ĐIỀU KHIỂN TƯƠNG TRANH

- **Sử dụng khóa (Locking)**: Khóa được dùng để sắp xếp thứ tự các giao tác theo thứ tự đi đến của chúng trên cùng một khoản mục dữ liệu.
- **Điều khiển tương tranh lạc quan (Optimistic concurrency control)**: Cho phép các thao tác được tiếp tục thực hiện cho đến khi sẵn sàng COMMIT và sau đó thực hiện kiểm tra xem có thao tác nào bị xung đột hay không.
- **Thứ tự nhãn thời gian**: Sử dụng nhãn thời gian để sắp thứ tự giao tác theo thời gian bắt đầu của chúng.

KHÓA LOẠI TRỪ

- Khóa loại trừ là cơ chế đơn giản để tuần tự hóa việc thực hiện giao tác.
- Phân loại:
  - Khóa hai pha (Two-phase): Pha thứ nhất giữ khóa, pha thứ hai mở khóa
  - Khóa hai pha nghiêm ngặt: Giữ khóa cho đến khi thực hiện COMMIT hoặc ABORT cho giao tác

KHÓA LOẠI TRỪ

Giao tác T		Giao tác U	
balance = b.getBalance() *b.setBalance(bal*1.1)* *a.withdraw(bal/10)*		*balance = b.getBalance()* *b.setBalance(bal*1.1)* *c.withdraw(bal/10)*	
Thao tác	Khóa	Thao tác	Khóa
openTransaction		*openTransaction*	
bal = b.getBalance()	lock *B*		
*b.setBalance(bal*1.1)*			
a.withdraw(bal/10)	lock *A*	*bal = b.getBalance()*	waits for *T*'s lock on *B*
closeTransaction	unlock *A*, *B*	• • •	
			lock *B*
		*b.setBalance(bal*1.1)*	
		c.withdraw(bal/10)	lock *C*
		closeTransaction	unlock *B*, *C*

KHÓA

- Các thao tác đọc trên cùng một khoản mục dữ liệu không gây nên xung đột.
- Khóa loại trừ giảm tính tương tranh hơn mức cần thiết
- Mô hình đọc nhiều/Ghi một (many reader/single write) phân biệt hai loại khóa: khóa chia sẻ và khóa ghi.
- Khóa hai pha hoặc hai pha chặt chẽ vẫn thường được dùng để đảm bảo tính tuần tự thực hiện

KHẢ NĂNG TƯƠNG THÍCH KHÓA

Cho một đối tượng		*Khóa yêu cầu*	
		read	*write*
Đã thiết lập khóa	*none*	OK	OK
	read	OK	wait
	write	wait	wait

QUẢN LÝ KHÓA

- Khi một thao tác truy nhập đối tượng bên trong giao tác:
  a) Nếu đối tượng chưa bị khóa, đối tượng sẽ được khóa và thao tác tiếp tục
  b) Nếu đã có giao tác khác khóa đối tượng và khóa đó xung đột với thao tác thì thao tác đó phải chờ cho đến khi mở khóa
  c) Nếu đã có giao tác khác khóa đối tượng và khóa đó không xung đột với thao tác thì khóa ở chế độ chia sẻ và thao tác tiếp tục
  d) Nếu đối tượng được khóa trong cùng một giao tác thì nâng mức khóa nếu cần thiết và thao tác tiếp tục (Ngăn chặn nâng mức khóa đối với khóa xung đột, sử dụng luật b)
- Khi thực hiện COMMIT hoặc ABORT, máy chủ sẽ mở khóa tất cả các đối tượng đã khóa cho giao tác

CÀI ĐẶT KHÓA

```
public class Lock
{
  private Object object; // Đối tượng đang được khóa
  private Vector holders;// Định danh giao tác giữ khóa
  private LockType lockType;  // Kiểu khóa
  public synchronized void acquire(TransID trans,LockType aLockType)
  {
    while(/*Giao tác khác giữ khóa trong chế độ xung đột*/)
    {
      try
      {
        wait();
      }
      catch (InterruptedException e)
      {
        /*Thực hiện thao tác xử lý lỗi*/
      }
    }
    if(holders.isEmpty())
    { // no TIDs  hold lock
      holders.addElement(trans);
      lockType  = aLockType;
    }
    else
    {
      if(/*Tiến hành khóa giữ khóa*/)
```

CÀI ĐẶT QUẢN LÝ KHÓA

```
public class LockManager
{
  private Hashtable theLocks;
  public  void setLock(Object object,TransID trans,LockType lockType)
  {
    Lock foundLock;
    synchronized(this)
    {
      // Tìm khóa liên quan đến đối tượng, nếu không thấy thì tạo khóa
         và thêm vào bảng băm
    }
    foundLock.acquire(trans,lockType);
  }
  // Đồng bộ khóa để đảm bảo loại bỏ tất cả các mục liên qan đến khóa
  public synchronized void unLock(TransID trans)
  {
    Enumeration e = theLocks.elements();
    while(e.hasMoreElements())
    {
      Lock aLock = (Lock)(e.nextElement());
      if(/* Nếu giao tác giữ khóa*/ )
      {
        aLock.release(trans);
      }
    }
  }
}
```

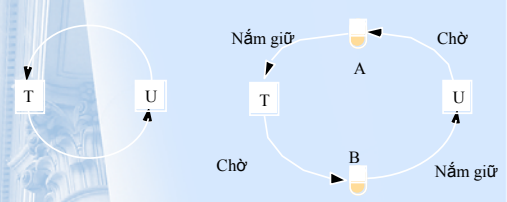
KHÓA CHẾT (Deadlocks)

- Sử dụng khóa có thể dẫn đến trạng thái *deadlock*.
- Deadlock là trạng thái, trong đó mỗi thành viên của nhóm các giao tác đang chờ thành viên khác giải phóng khóa
- Có thể sử dụng đồ thị *wait-for* để thể hiện các quan hệ chờ giữa các giao tác tương tranh tại máy chủ

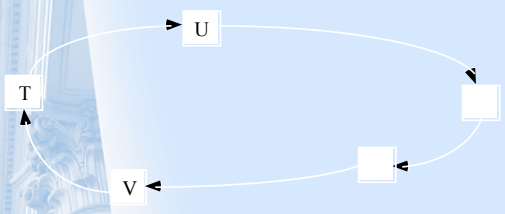
Deadlock với các khóa ghi (WRITE)

Giao tác T		Giao tác U	
Thao tác	Khóa	Thao tác	Khóa
a.deposit(100);	Khóa ghi A		
		b.deposit(200)	Khóa ghi B
b.withdraw(100)			
. . .	Chờ khóaU's trên B	a.withdraw(200);	Chờ khóa T's trên A
. . .		. . .	
. . .		. . .	
		. . .	

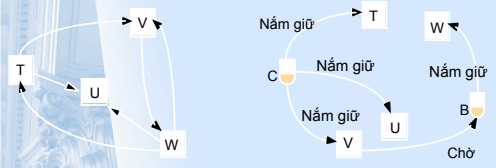
ĐỒ THỊ CHỜ KHÓA



CHU KỲ TRONG ĐỒ THỊ CHỜ KHÓA



ĐỒ THỊ CHỜ KHÓA



CÁC KỸ THUẬT PHÒNG NGỪA Deadlock

- Khi bắt đầu giao tác sử dụng khóa tất cả các mục dữ liệu
- Mỗi giao tác yêu cầu khóa trên các mục dữ liệu theo thứ tự đã định nghĩa trước
- Mỗi khóa có khoảng thời gian giới hạn, sau thời gian đó sẽ không được bảo vệ

PHÁT HIỆN Deadlock

- Có thể phát hiện Deadlock bằng cách tìm các chu kỳ trong đồ thị chờ khóa
- Hai vấn đề thiết kế:
  - Thường xuyên kiểm tra sự tồn tại của chu kỳ khóa
  - Chọn giao tác để hủy bỏ

XỬ LÝ Deadlock

Giao tác T		Giao tác U	
Thao tác	Khóa	Thao tác	Khóa
a.deposit(100);	Khóa ghi A		
		b.deposit(200)	Khóa ghi B
b.withdraw(100)			
• • •	Chờ khóa U's trên B	a.withdraw(200);	Chờ khóa T's trên A
	(Quá thời gian)	• • •	
Khóa T' trênA không còn được bảo vệ		• • •	
	mở khóaA, hủy T	a.withdraw(200);	Khóa ghi A trên A, B

TĂNG TÍNH TƯƠNG TRANH

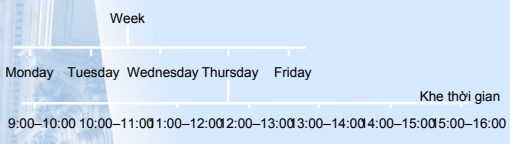
- **Khóa hai phiên bản**: Cho phép một giao tác tạm thời ghi dữ liệu trong khi các giao tác khác đọc dữ liệu chưa được COMMIT. Các thao tác đọc chỉ chờ nếu có giao tác khác đang COMMIT dữ liệu đó
- **Khóa có thứ bậc**: ở mỗi mức, việc thiết lập khóa cha có kết quả như đặt tất cả các khóa con tương đương

TƯƠNG THÍCH KHÓA

Cho một đối tượng		*Khóa cần đặt*		
		read	*write*	*commit*
Đã đặt khóa	*none*	OK	OK	OK
	read	OK	OK	wait
	write	OK	wait	
	commit	wait	wait	

KHÓA PHÂN CẤP

- Trong thực tế tồn tại các quan hệ cha con
- Ví dụ đơn giản: thời gian trong tuần



TƯƠNG THÍCH KHÓA PHÂN CẤP

Cho một đối tượng		*Khóa cần thiết lập*			
		read	*write*	*I-read*	*I-write*
Thiết lập khóa	*none*	OK	OK	OK	OK
	read	OK	wait	OK	wait
	write	wait	wait	wait	wait
	I-read	OK	wait	OK	OK
	I-write	wait	wait	OK	OK

NHƯỢC ĐIỂM CỦA CƠ CHẾ KHÓA

- Việc duy trì khóa tăng thêm tải xử lý, một số khóa có thể không cần thiết
- Giảm tính tương tranh để tránh deadlock hoặc giữ khóa cho tới khi kết thúc giao tác (tránh hủy bỏ theo tầng)

ĐIỀU KHIỂN TƯƠNG TRANH LẠC QUAN

- Các giao tác được phép tiếp tục nếu không có xung đột với các giao tác khác
- Khi phát hiện xung đột thì sẽ hủy bỏ giao tác nào đó.
- Mỗi giao tác gồm ba pha:
  - Pha đọc: sử dụng bản tạm thời cho mỗi mục dữ liệu được cập nhật
  - Pha phê chuẩn: Kiểm tra xem có xung đột hay không
  - Pha ghi: Nếu được phê chuẩn không có xung đột thì chuyển bản dữ liệu tạm thời thành vĩnh viễn

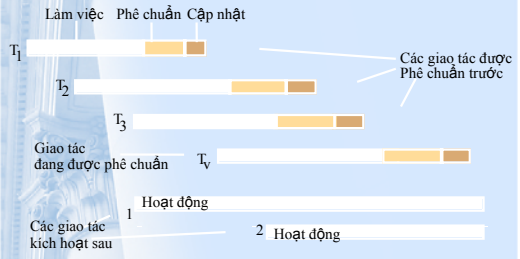
CÁC LUẬT ĐỌC/GHI

T_v	T_i	Luật	
write	read	1. T_i không được đọc đối tượng được	T_v ghi
read	write	2. T_i không được ghi đối tượng được	T_v đọc
write	write	3. T_i không được ghi đối tượng được	T_v ghi
		T_v không được ghi đối tượng được	T_i ghi

PHÊ CHUẨN GIAO TÁC

- Để được phê chuẩn, mỗi giao tác được gán số thứ tự (tăng dần) khi bước vào pha phê chuẩn
- Giao tác luôn luôn kết thúc pha đọc của mình sau tất cả các giao tác có số thứ tự thấp hơn.
- Các pha phê chuẩn có thể chồng nhau nhưng số hiệu phải được gán tuần tự.
- Tất cả các pha ghi được thực hiện tuần tự theo số hiệu đã được gán, như vậy không cần kiểm tra xung đột ghi-ghi
- Không nên tái sử dụng số thứ tự đã gán cho giao tác
- Hai dạng phê chuẩn:
  - Phê chuẩn ngược: Kiểm tra với các giao tác đã bước vào giai đoạn phê chuẩn trước nó.
  - Phê chuẩn xuôi: Kiểm tra với các giao tác sau nhưng vẫn đang hoạt động

PHÊ CHUẨN GIAO TÁC



PHÊ CHUẨN GIAO TÁC NGƯỢC

- Tất cả các thao tác đọc của các giao tác đã được thực hiện trước khi giao tác T_v bắt đầu không thể bị ảnh hưởng bởi thao tác ghi của T_v .
- Phê chuẩn giao tác của T_v chỉ cần kiểm tra tập đọc của T_v với tập ghi của các giao tác trước.
- startTn+1 là số hiệu giao tác lớn nhất đã được phê chuẩn, finishTn là số hiệu giao tác lớn nhất đã được gán tại thời điểm T_v bước vào giai đoạn phê chuẩn

```
boolean valid = true;
for (int Ti=startTn+1;Ti>=finishTn;Ti--)
{
 if (Tập đọc của Tv giao với tập ghi của Ti)
 {
  valid = false;
 }
}
```

PHÊ CHUẨN GIAO TÁC XUÔI

- Luật Ghi-Đọc đương nhiên thỏa mãn vì các giao tác được gán số hiệu sau chỉ được thực hiện sau khi giao tác trước đã kết thúc thao tác đọc.
- Tập ghi của Tv được so sánh với tập đọc của tất cả các giao tác vẫn đang hoạt động
- Giả sử sau giao thác Tv có các giao tác active1… activeN đang hoạt động

```
boolean valid = true;
for (int Tid = active1; Tid <= activeN; Tid++)
{
 if (tập ghi của Tv giao với tập đọc của Tid)
 {
  valid = false;
 }
}
```

SO SÁNH PHÊ CHUẨN XUÔI VÀ NGƯỢC

- Phê chuẩn ngược:
  - Bỏ qua các giao tác đang được phê chuẩn
  - Phải nhớ tập ghi của các giao tác đã COMMIT có thể xung đột với các giao tác đang hoạt động
  - So sánh tập đọc có thể rất lớn so với tập ghi cũ
- Phê chuẩn xuôi:
  - Ba lựa chọn:
    - Hoãn phê chuẩn cho đến khi giao tác xung đột (hoạt động) kết thúc
    - Hủy bỏ tất cả các giao tác đang xung đột và COMMIT giao tác đang được phê chuẩn
    - Hủy bỏ giao tác đang được phê chuẩn
  - Trong thời gian phê chuẩn, phải cho phép giao tác mới được phép bắt đầu
  - So sánh tập ghi nhỏ với tập đọc của các giao tác đang hoạt động

THỨ TỰ NHÃN THỜI GIAN

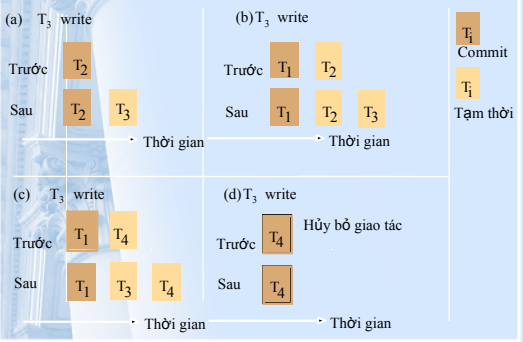
- Mỗi giao tác được gán nhãn thời gian duy nhất khi bắt đầu
- Mỗi thao tác mang nhãn thời gian của giao tác đã được cấp phát và được phê chuẩn khi thực hiện.
- Nếu thao tác không được thực hiện thì giao tác sẽ bị hủy bỏ ngay lập tức.

LUẬT ĐỌC/GHI

Luật	T_c	T_i	
1.	write	read	T_c không được ghi đối tượng đang được T_i đọc trong đó $T_i>T_c$ điều này yêu cầu $T_c \geq$ nhãn thời gian đọc lớn nhất của đối tượng.
2.	write	write	T_c không được ghi đối tượng đang được T_i ghi trong đó $T_i>T_c$ điều này yêu cầu $T_c \geq$ Nhãn thời gian ghi của đối tượng đã COMMIT
3.	read	write	T_c không được đọc đối tượng đang được T_i ghi trong đó $T_i>T_c$ điều này yêu cầu $T_c >$ Nhãn thời gian ghi của đối tượng đã COMMIT

Nguyễn Xuân Anh, anhcz@yahoo.com

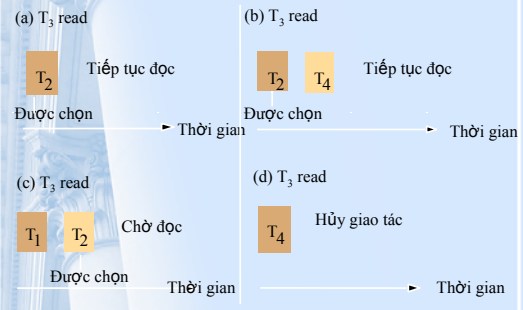
THAO TÁC GHI VÀ NHÃN THỜI GIAN



THAO TÁC GHI VÀ NHÃN THỜI GIAN

```
if (Tc ≥ nhãn thời gian đọc lớn nhất trên D &&
    Tc > nhãn thời gian ghi trên bản đã COMMIT
         của D)
{
  Thực hiện ghi trên bản tạm thời của D với nhãn
  thời gian Tc
}
else
{
  Hủy giao tác Tc
}
```

THAO TÁC ĐỌC VÀ NHÃN THỜI GIAN



THAO TÁC ĐỌC VÀ NHÃN THỜI GIAN

```
if (Tc>Nhãn thời gian GHI trên bản COMMIT của D)
{
  Giả thiết D được chọn là phiên bản của D với nhãn
  thời gian GHI cao nhất ≤ Tc
  if (D được chọn đã được COMMIT)
  {
    Thực hiện thao tác đọc trên bản D đã chọn
  }
  else
  {
    Chờ cho đến khi giao tác thực hiện xong
    việc COMMIT hoặc ABORT bản D đã chọn sau đó áp
    dụng lại luật đọc
  }
}
Else
{
  Hủy giao tác Tc
}
```

THỨ TỰ NHÃN THỜI GIAN

		Nhãn thời gian và phiên bản đối tượng					
T	U	A		B		C	
		RTS {}	WTS S	RTS {}	WTS S	RTS {}	WTS S
openTransaction							
bal = b.getBalance()				{T}			
	openTransaction						
b.setBalance(bal*1.1)					S, T		
	bal = b.getBalance()						
	wait for T						
a.withdraw(bal/10)	• • •		S, T				
commit	• • •		T		T		
	bal = b.getBalance()			{U}			
	b.setBalance(bal*1.1)				S, U		
	c.withdraw(bal/10)						S, U

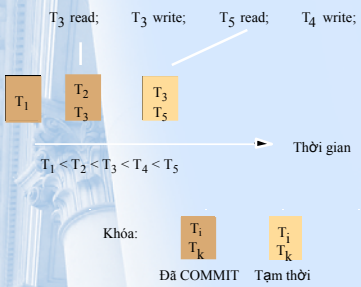
LUẬT NỚI LỎNG GHI
(Relaxing the Write Rule)

- Nếu lệnh GHI quá trễ thì có thể bị bỏ qua thay cho việc hủy bỏ giao tác, bởi vì nếu nó đến đúng hạn thì kết quả của nó cũng sẽ bị ghi đè
- Tuy nhiên nếu giao tác khác đọc khoản mục dữ liệu đó thì giao tác với lện GHI trễ sẽ hỏng bởi vì nhãn thời gian ĐỌC trên khoản mục dữ liệu này.

THỨ TỰ NHÃN THỜI GIAN ĐA PHIÊN BẢN

- Máy chủ giữ các phiên bản đã COMMIT cũ cũng như các phiên phản tạm thời trong danh sách các phiên bản của các khoản mục dữ liệu
- Với danh sách đó, các thao tác ĐỌC đến quá chậm sẽ không bị từ chối.
- Thao tác ĐỌC của giao tác được chỉ định đến phiên bản với nhãn thời gian ghi lớn nhất và nhỏ hơn nhãn thời gian của giao tác.

LỆNH GHI MUỘN KHÔNG PHÊ CHUẨN LỆNH ĐỌC



SO SÁNH ĐIỀU KHIỂN TƯƠNG TRANH

Mục	Khóa	Nhãn thời gian
Tính chất	Bi quan	Lạc quan
Thứ tự quyết định	Động	Tĩnh
Giao tác được lợi	Ghi nhiều hơn đọc	Chỉ đọc
Giải quyết xung đột	Chờ, hủy bỏ	Hủy bỏ