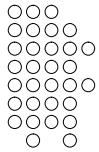
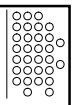
Chương 2

Điều khiển đồng thời

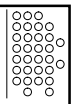


Nội dung chi tiết



- Các vấn đề trong truy xuất đồng thời
  - Mất dữ liệu đã cập nhật (lost updated)
  - Không thể đọc lại (unrepeatable read)
  - "Bóng ma" (phantom)
  - Đọc dữ liệu chưa chính xác (dirty read)
- Kỹ thuật khóa (locking)
  - Giới thiệu
  - Khóa 2 giai đoạn (two-phase)
  - Khóa đọc viết
  - Khóa đa hạt (multiple granularity)
  - Nghi thức cây (tree protocol)

Nội dung chi tiết (tt)



- Kỹ thuật nhãn thời gian (timestamps)
  - Giới thiệu
  - Nhãn thời gian toàn phần
  - Nhãn thời gian riêng phần
  - Nhãn thời gian nhiều phiên bản (multiversion)
- Kỹ thuật xác nhận hợp lệ (validation)

Vđề mất dữ liệu đã cập nhật

- Xét 2 giao tác

T_1	T_2
Read(A)	Read(A)
A:=A+10	A:=A+20
Write(A)	Write(A)

- Giả sử T_1 và T_2 được thực hiện đồng thời
  - Dữ liệu đã cập nhật tại t_4 của T_1 bị mất vì đã bị ghi chồng lên ở thời điểm t_6

A=50	T_1	T_2
t_1	Read(A)	
t_2		Read(A)
t_3	A:=A+10	
t_4	**Write(A)**	
t_5		A:=A+20
t_6		**Write(A)**
	A=60	A=70

Vđề không thể đọc lại

- Xét 2 giao tác

T_1	T_2
Read(A)	Read(A)
A:=A+10	Print(A)
Write(A)	Read(A)
	Print(A)

- Giả sử T_1 và T_2 được thực hiện đồng thời
  - T_2 tiến hành đọc A hai lần thì cho hai kết quả khác nhau

A=50	T_1	T_2	
t_1	Read(A)		
t_2		**Read(A)**	A=50
t_3	A:=A+10		
t_4		Print(A)	A=50
t_5	Write(A)		
t_6		**Read(A)**	A=60
t_7		Print(A)	A=60

Vđề "bóng ma"

- Xét 2 giao tác T_1 và T_2 được xử lý đồng thời
  - A và B là 2 tài khoản
  - T_1 rút 1 số tiền ở tài khoản A rồi đưa vào tài khoản B
  - T_2 kiểm tra đã nhận đủ tiền hay chưa?

A=70, B=50	T_1	T_2		
t_1	Read(A)		A=70	
t_2	A:=A-50			
t_3	**Write(A)**		A=20	
t_4		Read(A)	A=20	
t_5		Read(B)	B=50	
t_6		**Print(A+B)**	A+B=70	mất 50 ???
t_7	Read(B)			
t_8	B:=B+50			
t_9	**Write(B)**			

Vđề đọc dữ liệu chưa chính xác

- Xét 2 giao tác T_1 và T_2 được xử lý đồng thời
  - T_2 đã đọc dữ liệu được ghi bởi T_1 nhưng sau đó T_1 yêu cầu hủy việc ghi

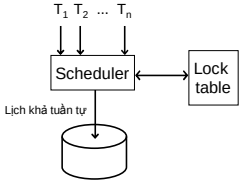
	T_1	T_2
t_1	Read(A)	
t_2	A:=A+10	
t_3	Write(A)	
t_4		**Read(A)**
t_5		Print(A)
t_6	**Abort**	

Nội dung chi tiết

- Các vấn đề truy xuất đồng thời
- Kỹ thuật khóa (lock)
  - Giới thiệu
  - Khóa 2 giai đoạn (two-phase)
  - Khóa đọc viết
  - Khóa đa hạt (multiple granularity)
  - Nghi thức cây (tree protocol)
- Kỹ thuật nhãn thời gian (timestamp)
- Kỹ thuật xác nhận tính hợp lệ (validation)

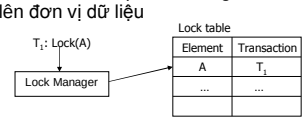
Giới thiệu

- Làm thế nào để bộ lập lịch ép buộc 1 lịch phải khả tuần tự?
- Bộ lập lịch với cơ chế khóa (locking scheduler)
  - Có thêm 2 hành động
    - Lock
    - Unlock



Kỹ thuật khóa

- Các giao tác trước khi muốn đọc/viết lên 1 đơn vị dữ liệu phải phát ra 1 yêu cầu xin khóa (lock) đơn vị dữ liệu đó
  - Lock(A) hay l(A)
- Yêu cầu này được bộ phận quản lý khóa xử lý
  - Nếu yêu cầu được chấp thuận thì giao tác mới được phép đọc/ghi lên đơn vị dữ liệu



- Sau khi thao tác xong thì giao tác phải phát ra lệnh giải phóng đơn vị dữ liệu (unlock)
  - Unlock(A) hay u(A)

Kỹ thuật khóa (tt)

- Qui tắc
  - (1) Giao tác đúng đắn

T_i : … l(A) … r(A) / w(A) … u(A) …

  - (2) Lịch thao tác hợp lệ

S : … l_i(A) …… u_i(A) …

không có l_j(A)

Ví dụ

S

T_1	T_2
Lock(A)	
Read(A,t)	
t:=t+100	
Write(A,t)	
Unlock(A)	
	Lock(A)
	Read(A,s)
	s:=s*2
	Write(A,s)
	Unlock(A)
	Lock(B)
	Read(B,s)
	s:=s*2
	Write(B,s)
	Unlock(B)
Lock(B)	
Read(B,t)	
t:=t+100	
Write(B,t)	
Unlock(B)	

Bài tập

- Cho biết lịch nào hợp lệ? Giao tác nào là đúng?

S_1

T_1	T_2	T_3
Lock(A)		
Lock(B)		
Read(A)		
Write(B)		
	Lock(B)	
Unlock(A)		
Unlock(B)		
	Read(B)	
	Write(B)	
	Unlock(B)	
		Lock(B)
		Read(B)
		Unlock(B)

S_2

T_1	T_2	T_3
Lock(A)		
Read(A)		
Write(B)		
Unlock(A)		
Unlock(B)		
	Lock(B)	
	Read(B)	
	Write(B)	
		Lock(B)
		Read(B)
		Unlock(B)

Bài tập (tt)

- Cho biết lịch nào hợp lệ? Giao tác nào là đúng?

S_3

T_1	T_2	T_3
Lock(A)		
Read(A)		
Unlock(A)		
Lock(B)		
Write(B)		
Unlock(B)		
	Lock(B)	
	Read(B)	
	Write(B)	
	Unlock(B)	
		Lock(B)
		Read(B)
		Unlock(B)

Kỹ thuật khóa (tt)

- Nếu lịch S hợp lệ thì S có khả tuần tự không?

S

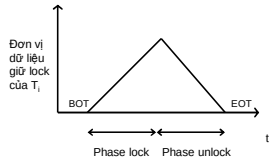
T_1	T_2	A	B
		25	25
Lock(A); Read(A,t)			
t:=t+100			
Write(A,t); **Unlock(A)**		125	
	Lock(A); Read(A,s)		
	s:=s*2		
	Write(A,s); **Unlock(A)**	**250**	
	Lock(B); Read(B,s)		
	s:=s*2		
	Write(B,s); **Unlock(B)**		50
Lock(B); Read(B,t)			
t:=t+100			
Write(B,t); **Unlock(B)**			**150**

Kỹ thuật khóa 2 giai đoạn (2PL)

- Qui tắc
  - (3) Giao tác 2PL

S : …… $l_i(A)$ …… $u_i(A)$ …

không có unlock — **không có lock**



Thực hiện xong hết tất cả các yêu cầu lock rồi mới tiến hành unlock

Kỹ thuật khóa 2 giai đoạn (tt)

T_1
Lock(A)
Read(A)
Lock(B)
Read(B)
B:=B+A
Write(B)
Unlock(A)
Unlock(B)

T_2
Lock(B)
Read(B)
Lock(A)
Read(A)
Unlock(B)
A:=A+B
Write(A)
Unlock(A)

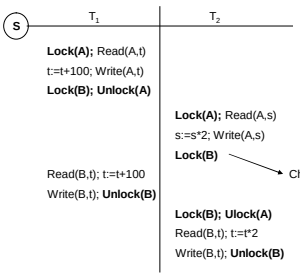
Thỏa nghi thức khóa 2 giai đoạn

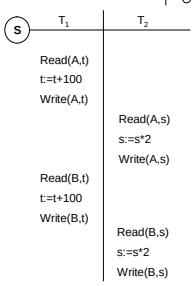
T_3
Lock(B)
Read(B)
B=B-50
Write(B)
Unlock(B)
Lock(A)
Read(A)
A=A+50
Write(A)
Unlock(A)

T_4
Lock(A)
Read(A)
Unlock(A)
Lock(B)
Read(B)
Unlock(B)
Pritn(A+B)

Không thỏa nghi thức khóa 2 giai đoạn

Kỹ thuật khóa 2 giai đoạn (tt)



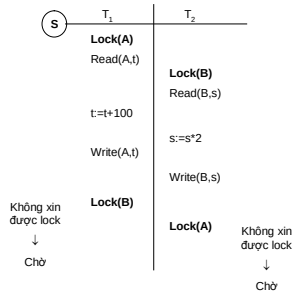


Kỹ thuật khóa 2 giai đoạn (tt)

- Định lý
  - S thỏa qui tắc (1), (2), (3) $\Rightarrow$ S conflict- serializable
- Chứng minh
  - Giả sử G(S) có chu trình
  - $T_1 \rightarrow T_2 \rightarrow \ldots T_n \rightarrow T_1$
  - T_1 thực hiện lock những đơn vị dữ liệu được unlock bởi T_n
  - T_1 có dạng … lock … unlock … lock
  - Điều này vô lý vì T_1 là giao tác thỏa 2PL
  - G(S) không thể có chu trình
  - S conflict-serializable

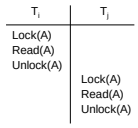
Kỹ thuật khóa 2 giai đoạn (tt)

- Chú ý



Kỹ thuật khóa đọc viết

- Vấn đề



- Bộ lập lịch có các hành động
  - Khóa đọc (Read lock, Shared lock)
    - RLock(A) hay rl(A)
  - Khóa ghi (Write lock, Exclusive lock)
    - WLock(A) hay wl(A)
  - Giải phóng khóa
    - Unlock(A) hay u(A)

Kỹ thuật khóa đọc viết (tt)

- Cho 1 đơn vị dữ liệu A bất kỳ
  - WLock(A)
    - Hoặc có 1 khóa ghi duy nhất lên A
    - Hoặc không có khóa ghi nào lên A
  - RLock(A)
    - Có thể có nhiều khóa đọc được thiết lập lên A

Kỹ thuật khóa đọc viết (tt)

- Giao tác muốn **Write(A)**
  - Yêu cầu WLock(A)
  - WLock(A) sẽ được chấp thuận nếu A tự do
    - Sẽ không có giao tác nào nhận được WLock(A) hay RLock(A)
- Giao tác muốn **Read(A)**
  - Yêu cầu RLock(A) hoặc WLock(A)
  - RLock(A) sẽ được chấp thuận nếu A không đang giữ một WLock nào
    - Không ngăn chặn các thao tác khác cùng xin Rlock(A)
    - Các giao tác không cần phải chờ nhau khi đọc A
- Sau khi thao tác xong thì giao tác phải giải phóng khóa trên đơn vi dữ liệu A
  - ULock(A)

Kỹ thuật khóa đọc viết (tt)

- Qui tắc
  - (1) Giao tác đúng đắn

T_i : … rl(A) … r(A) … u(A) …

T_i : … wl(A) … w(A) … u(A) …

Kỹ thuật khóa đọc viết (tt)

- Vấn đề
  - Các giao tác đọc và ghi trên cùng 1 đơn vị dữ liệu

- Giải quyết
  - Cách 1 - yêu cầu khóa độc quyền

    T_i : … wl(A) … r(A) … w(A) … u(A) …

  - Cách 2 - nâng cấp khóa

    T_i : … rl(A) … r(A) … wl(A) … w(A) … u(A) …

Bài tập

- Hãy suy nghĩ và cho biết cách nào là hợp lý
  - Xin khóa thứ 2 cho đơn vị dữ liệu muốn ghi?
  - Xin khóa độc quyền ngay từ đầu?
- Cho ví dụ và giải thích

Kỹ thuật khóa đọc viết (tt)

- Qui tắc
  - (2) - Lịch thao tác hợp lệ

S : … $rl_i(A)$ …… $u_i(A)$ …

không có $wl_j(A)$

S : … $wl_i(A)$ …… $u_i(A)$ …

không có $wl_j(A)$
không có $rl_j(A)$

Kỹ thuật khóa đọc viết (tt)

- Ma trận tương thích (compatibility matrices)

Trạng thái hiện hành	Yêu cầu lock: Share	Yêu cầu lock: eXclusive
Share	yes	no
eXclusive	no	no

Kỹ thuật khóa đọc viết (tt)

- Qui tắc
  - (3) - Giao tác 2PL
    - Ngoại trừ trường hợp nâng cấp khóa, các trường hợp còn lại đều giống với nghi thức khóa
    - Nâng cấp xin nhiều khóa hơn

      S : … $rl_i(A)$ … $wl_i(A)$ …… $u_i(A)$ …

      không có unlock ← **không có lock** →
    - Nâng cấp giải phóng khóa đọc

      S : … $rl_i(A)$ … $ul_i(A)$ … $wl_i(A)$ …… $u_i(A)$ …

      vẫn chấp nhận trong pha lock ←

Kỹ thuật khóa đọc viết (tt)

- Định lý
  - S thỏa qui tắc (1), (2), (3) ⇒ S conflic-serializable của khóa đọc viết
- Chứng minh
  - Bài tập về nhà

Ví dụ

S

T_1	T_2
RL(A)	
Read(A)	
UL(A)	
	RL(B)
	Read(B)
	UL(B)
	WL(A)
	Read(A)
	A:=A+B
	Write(A)
	UL(A)
WL(B)	
Read(B)	
B:=B+A	
Write(B)	
UL(B)	

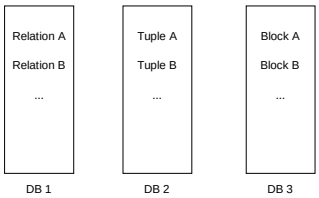
- S có khả tuần tự không?
- Giao tác nào không thỏa 2PL?

Ví dụ (tt)

T_1	T_2	T_3	T_4
	RL(A)		
		RL(A)	
	WL(B)		
	UL(A)		
		WL(A)	
	UL(B)		
RL(B)			
		UL(A)	
			RL(B)
RL(A)			
			UL(B)
WL(C)			
UL(A)			
			WL(B)
			UL(B)
UL(B)			
UL(C)			

- S có khả tuần tự hay không?
- Giao tác nào không thỏa 2PL?

Khóa ở mức độ nào?

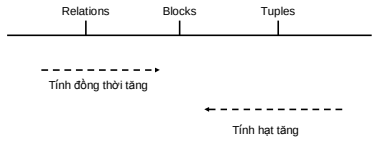


Khóa ở mức độ nào? (tt)

- Xét ví dụ hệ thống ngân hàng
  - Quan hệ TàiKhoản(mãTK, sốDư)
  - Giao tác gửi tiền và rút tiền
    - Khóa relation?
      - Các giao tác thay đổi giá trị của sốDư nên yêu cầu khóa độc quyền
      - Tại 1 thời điểm chỉ có hoặc là rút hoặc là gửi
      - Xử lý đồng thời chậm
    - Khóa tuple hay disk block?
      - 2 tài khoản ở 2 blocks khác nhau có thể được cập nhật cùng thời điểm
      - Xử lý đồng thời nhanh
  - Giao tác tính tổng số tiền của các tài khoản
    - Khóa relation?
    - Khóa tuple hay disk block?

Khóa ở mức độ nào? (tt)

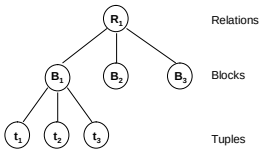
- Phải quản lý khóa ở nhiều mức độ
  - Tính chất hạt (granularity)
  - Tính đồng thời (concurrency)



Có thể có cả 2 tính không?

Phân cấp dữ liệu

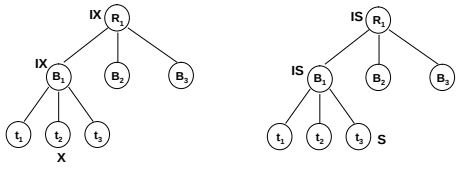
- Relations là đơn vị dữ liệu khóa lớn nhất
- Một relation gồm 1 hoặc nhiều blocks (pages)
- Một block gồm 1 hoặc nhiều typles



Kỹ thuật khóa đa hạt

- Gồm các khóa
  - Khóa thông thường
    - Shared lock: S
    - Exclusive lock: X
  - Khóa cảnh báo (warning lock)
    - Warning (intention to) shared lock: IS
    - Warning (intention to) exclusive lock: IX

Kỹ thuật khóa đa hạt (tt)



Kỹ thuật khóa đa hạt (tt)

Trạng thái hiện hành	Yêu cầu lock: IS	IX	S	X
IS	yes	yes	yes	no
IX	yes	yes	no	no
S	yes	no	yes	no
X	no	no	no	no

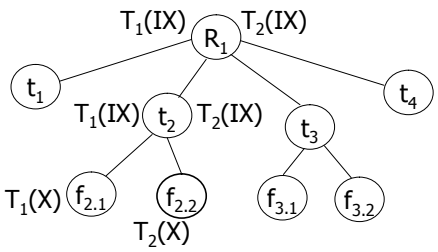
Kỹ thuật khóa đa hạt (tt)

Nút cha đã khóa bằng phương thức	Nút con có thể khóa bằng các phương thức
IS	**IS, S**
IX	**IS, S, IX, X**
S	**[S, IS] không cần thiết**
X	**Không có**

Kỹ thuật khóa đa hạt (tt)

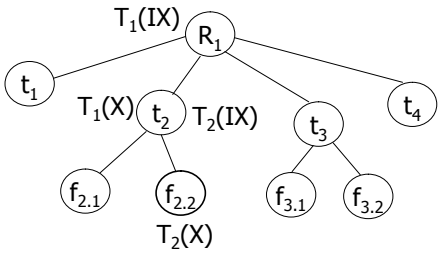
- (1) Thỏa ma trận tương thích
- (2) Khóa nút gốc của cây trước
- (3) Nút Q có thể được khóa bởi T_i bằng S hay IS khi cha(Q) đã bị khóa bởi T_i bằng IX hay IS
- (4) Nút Q có thể được khóa bởi T_i bằng X hay IX khi cha(Q) đã bị khóa bởi T_i bằng IX
- (5) T_i thỏa 2PL
- (6) T_i có thể giải phóng nút Q khi không có nút con nào của Q bị khóa bởi T_i

Bài tập



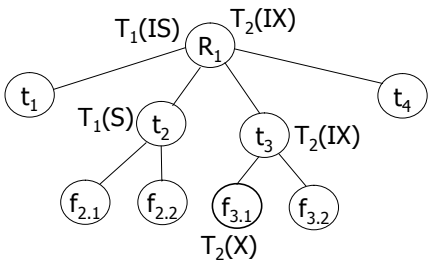
- T_2 có thể truy xuất $f_{2.2}$ bằng khóa X được không?
- T_2 sẽ có những khóa gì?

Bài tập (tt)



- T_2 có thể truy xuất $f_{2.2}$ bằng khóa X được không?
- T_2 sẽ có những khóa gì?

Bài tập (tt)



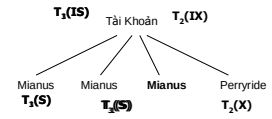
- T_2 có thể truy xuất $f_{3.1}$ bằng khóa X được không?
- T_2 sẽ có những khóa gì?

Kỹ thuật khóa đa hạt (tt)

- Vấn đề

Tài Khoản

MãTK	SốDư	MãChiNhánh
A-101	500	Mianus
A-215	700	Mianus
A-102	400	Perryride
A-201	900	Mianus



T_1
```
select * from TaiKhoan
where maCN="Mianus"
```

T_2
```
update TaiKhoan
set soDu=400
where maCN="Perryride"
```

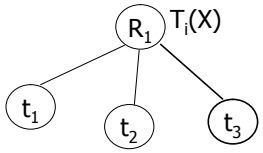
T_3
```
select sum(soDu)
from TaiKhoan
where maCN="Minanus"
```

T_4
```
insert TaiKhoan
valuse(A-201, 900, Mianus )
```

T_3 có còn đúng không?

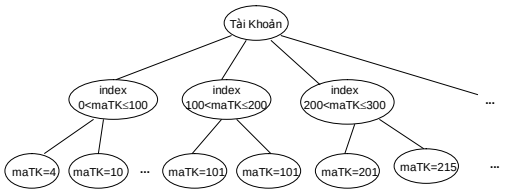
Kỹ thuật khóa đa hạt (tt)

- Giải pháp
  - Trước khi thêm vào 1 nút Q ta phải khóa cha(Q) bằng khóa X



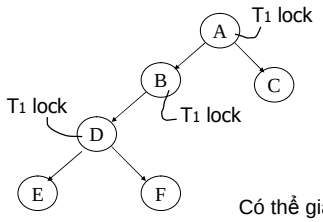
Nghi thức cây

- Chỉ mục B-tree



Nghi thức cây (tt)

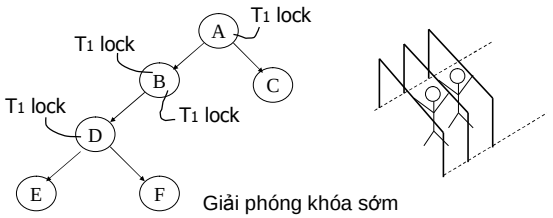
- Muốn truy xuất nút D thì phải duyệt qua nút cha(D) theo chiều của con trỏ



Có thể giải phóng khóa tại nút A khi không cần khóa A nữa ???

Nghi thức cây (tt)

- Di chuyển như "vận động viên biểu diễn xà kép"



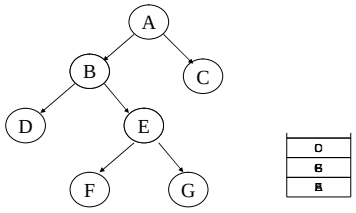
Giải phóng khóa sớm
→ không thỏa 2PL
→ lịch thao tác khả tuần tự ???

Nghi thức cây (tt)

- Giả sử
  - Dùng 1 khóa độc quyền: $Lock_i(X)$ hay $l_i(X)$
  - Các giao tác đúng đắn
  - Lịch thao tác hợp lệ
- Qui tắc
  - (1) Giao tác T_i có thể phát ra khóa đầu tiên ở bất kỳ nút nào
  - (2) Nút Q sẽ được khóa bởi T_i khi cha(Q) cũng được khóa bởi T_i
  - (3) Các nút có thể được giải phóng khóa bất cứ lúc nào
  - (4) Sau khi Ti giải phóng khóa trên Q, Ti không được khóa trên Q nữa

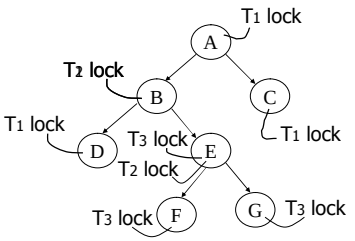
Ví dụ

T_1: l(A); r(A); l(B); r(B); l(C); r(C); w(A); u(A); l(D); r(D); w(B); u(B); w(D); u(D); w(C); u(C)
T_2: l(B); r(B); l(E); r(E); w(B); u(B); w(E); u(E)
T_3: l(E); r(E); l(F); r(F); w(F); u(F); l(G); r(G); w(E); u(E); w(G); u(G)



Ví dụ (tt)

T_1	T_2	T_3
L(A); R(A)		
L(B); R(B)		
L(C); R(C)		
W(A); U(A)		
L(D); R(D)		
W(B); U(B)		
	L(B); R(B)	
		L(E); R(E)
W(D); U(D)		
W(C); U(C)		
	L(E)	
	↓	
	Chờ	L(F); R(F)
		W(F); U(F)
		L(G); R(G)
		W(E); U(E)
	L(E); R(E)	
		W(G); U(G)
	W(B); U(B)	
	W(E); U(E)	



Lịch khả tuần tự theo nghi thức cây

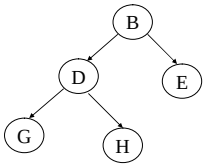
Ví dụ (tt)

T_1: l(B); l(E); l(D); u(B); u(E); l(G); u(D); u(G)

T_2: l(D); l(H); u(D); u(H)

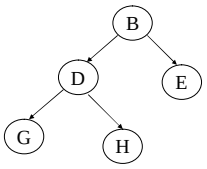
T_3: l(B); l(E); u(E); l(B)

T_4: l(D); l(H); u(D); u(H)



Ví dụ (tt)

T_1	T_2	T_3	T_4
Lock(B)			
	Lock(D)		
	Lock(H)		
	Unlock(D)		
Lock(E)			
Lock(D)			
Unlock(B)			
Unlock(E)			
		Lock(B)	
		Lock(E)	
	Unlock(H)		
Lock(G)			
Unlock(D)			
			Lock(D)
			Lock(H)
			Unlock(D)
			Unlock(H)
		Unlock(E)	
		Unlock(B)	
Unlock(G)			



Lịch khả tuần tự theo nghi thức cây không?

Nội dung chi tiết

- Các vấn đề truy xuất đồng thời
- Kỹ thuật khóa (locking)
- Kỹ thuật nhãn thời gian (timestamps)
  - Giới thiệu
  - Nhãn thời gian toàn phần
  - Nhãn thời gian riêng phần
  - Nhãn thời gian nhiều phiên bản (multiversion)
- Kỹ thuật xác nhận hợp lệ (validation)

Giới thiệu

- Ý tưởng
  - Giả sử không có hành động nào vi phạm tính khả tuần tự
  - Nhưng nếu có, hủy giao tác có hành động đó và thực hiện lại giao tác
- Chọn một thứ tự thực hiện nào đó cho các giao tác bằng cách gán nhãn thời gian (timestamping)
  - Mỗi giao tác T sẽ có 1 nhãn, ký hiệu TS(T)
    - Tại thời điểm giao tác bắt đầu
  - Thứ tự của các nhãn tăng dần
    - Giao tác bắt đầu trễ thì sẽ có nhãn thời gian lớn hơn các giao tác bắt đầu sớm

Giới thiệu (tt)

- Để gán nhãn
  - Đồng hồ của máy tính
  - Bộ đếm (do bộ lập lịch quản lý)
- Chiến lược cơ bản
  - Nếu $ST(T_i) < ST(T_j)$ thì lịch thao tác được phát sinh phải tương đương với lịch biểu tuần tự $\{T_i, T_j\}$

Nhãn thời gian toàn phần

- Mỗi giao tác T khi phát sinh sẽ được gán 1 nhãn TS(T) ghi nhận lại thời điểm phát sinh của T
- Mỗi đơn vị dữ liệu X cũng có 1 nhãn thời TS(X), nhãn này ghi lại TS(T) của giao tác T đã thao tác read/write thành công sau cùng lên X
- Khi đến lượt giao tác T thao tác trên dữ liệu X, so sánh TS(T) và TS(X)

Nhãn thời gian toàn phần (tt)

Read(T, X)

```
If TS(X) <= TS(T)
      Read(X);
      //cho phép đọc X
      TS(X):= TS(T);
Else
      Abort {T};
      //hủy bỏ T
      //khởi tạo lại ST
```

Write(T, X)

```
If TS(X) <= TS(T)
      Write(X);
      //cho phép ghi X
      TS(X):= TS(T);
Else
      Abort {T};
      //hủy bỏ T
      //khởi tạo lại TS
```


Ví dụ

	T_1	T_2	A	B	
	$TS(T_1)$=100	$TS(T_2)$=200	TS(A)=0	TS(B)=0	
1	Read(A)		TS(A)=100		TS(A) <= $TS(T_1)$: T_1 đọc được A
2		Read(B)		TS(B)=200	TS(B) <= $TS(T_2)$: T_2 đọc được B
	A=A*2				
3	Write(A)		TS(A)=100		TS(A) <= $TS(T_1)$: T_1 ghi lên A được
		B=B+20			
4		Write(B)		TS(B)=200	TS(B) <= $TS(T_2)$: T_2 ghi lên B được
5	Read(B)				TS(B) > $TS(T_1)$: T_1 không đọc được B

Abort

Khởi tạo lại $TS(T_1) \rightarrow TS(T_2) < TS(T_1)$

Lịch khả tuần tự theo thứ tự $\{T_2, T_1\}$

Nhãn thời gian toàn phần (tt)

- Nhận xét

T_1	T_2	A
$TS(T_1)=100$	$TS(T_2)=120$	$TS(A)=0$
Read(A)		$TS(A)=100$
	Read(A)	$TS(A)=120$
	Write(A)	$TS(A)=120$
Write(A)		

T_1 bị hủy và bắt đầu thực hiện lại với timestamp mới

T_1	T_2	A
$TS(T_1)=100$	$TS(T_2)=120$	$TS(A)=0$
Read(A)		$TS(A)=100$
	Read(A)	$TS(A)=120$
	Read(A)	$TS(A)=120$
Read(A)		

T_1 bị hủy và bắt đầu thực hiện lại với timestamp mới

Không phân biệt tính chất của thao tác là đọc hay viết
$\rightarrow T_1$ vẫn bị hủy và làm lại từ đầu với 1 timestamp mới

Nhãn thời gian riêng phần

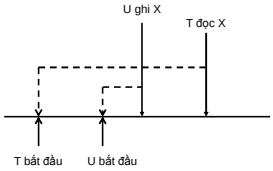
- Nhãn của đơn vị dữ liệu X được tách ra thành 2
  - RT(X) - read
    - Ghi nhận TS(T) gần nhất đọc X thành công
  - WT(X) - write
    - Ghi nhận TS(T) gần nhất ghi X thành công

Nhãn thời gian riêng phần (tt)

- Công việc của bộ lập lịch
  - Gán nhãn RT(X), WT(X) và C(X)
  - Kiểm tra thao tác đọc/ghi xuất hiện vào lúc nào
  - Có xãy ra tình huống
    - Đọc quá trễ
    - Ghi quá trễ
    - Đọc dữ liệu rác (dirty read)
    - Qui tắc ghi Thomas

Nhãn thời gian riêng phần (tt)

- Đọc quá trễ

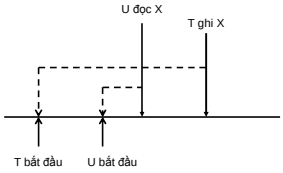


- ST(T) < ST(U)
- U ghi X trước, T đọc X sau
  - ST(T) < WT(X)
- T không thể đọc X sau U

→ Hủy T

Nhãn thời gian riêng phần (tt)

- Ghi quá trễ

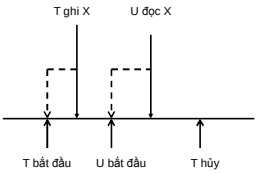


- ST(T) < ST(U)
- U đọc X trước, T ghi X sau
  - WT(X) < ST(T) < RT(X)
- U phải đọc được giá trị X được ghi bởi T

→ Hủy T

Nhãn thời gian riêng phần (tt)

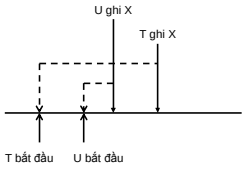
- Đọc dữ liệu rác



- ST(T) < ST(U)
- T ghi X trước, U đọc X sau
  - Thao tác bình thường
- Nhưng T hủy
  - U đọc X sau khi T commit
  - U đọc X sau khi T abort

Nhãn thời gian riêng phần (tt)

- Qui tắc ghi Thomas

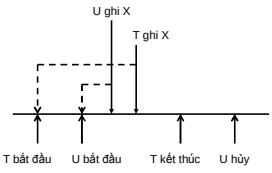


- ST(T) < ST(U)
- U ghi X trước, T ghi X sau
  - ST(T) < WT(X)
- T ghi X xong thì không làm được gì
  - Không có giao tác nào đọc được giá trị X của T (nếu có cũng bị hủy do đọc quá trễ)
  - Các giao tác đọc sau T và U thì mong muốn đọc giá trị X của U

→Bỏ qua thao tác ghi của T

Nhãn thời gian riêng phần (tt)

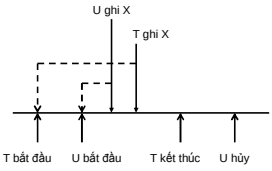
- Qui tắc ghi Thomas



- Nhưng U hủy
  - Giá trị của X được ghi bởi U bị mất
  - Cần khôi phục lại giá trị X trước đó
- Và T đã kết thúc
  - X có thể khôi phục từ thao tác ghi của T
- Do qui tắc ghi Thomas
  - Thao tác ghi đã được bỏ qua
  - Quá trễ để khôi phục X

Nhãn thời gian riêng phần (tt)

- Qui tắc ghi Thomas



- Khi T muốn ghi
  - Cho T thử ghi và sẽ gỡ bỏ nếu T hủy
  - Sao lưu giá trị cũ của X và nhãn WT(X) trước đó

Nhãn thời gian riêng phần (tt)

- Tóm lại
  - Khi có yêu cầu đọc và ghi từ giao tác T
  - Bộ lập lịch sẽ
    - Đáp ứng yêu cầu
    - Hủy T và khởi tạo lại T với 1 timestamp mới
      - T rollback
    - Trì hoãn T, sau đó mới quyết định phải hủy T hoặc đáp ứng yêu cầu

Nhãn thời gian riêng phần (tt)

Read(T, X)

```
If WS(X) <= TS(T)
      Read(X);//cho phép đọc X
      RT(X):= max(RT(X),TS(T));
Else
      Rollback{T};
      //hủy T và khởi tạo lại TS mới
```

Write(T, X)

```
If RT(X) <= TS(T)
      If WT(X) <= TS(T)
            Write(X);//cho phép ghi X
            WT(X):= TS(T);
      //Else không làm gì cả
Else
      Rollback{T};
      //hủy T và khởi tạo lại TS mới
```


Ví dụ

	T_1 $TS(T_1)=100$	T_2 $TS(T_2)=200$	A RT(A)=0 WT(A)=0	B RT(B)=0 WT(B)=0	C RT(C)=0 WT(C)=0	
1	Read(A)		RT(A)=100 WT(A)=0			$WT(A) < TS(T_1)$ T_1 đọc được A
2		Read(B)		RT(B)=200 WT(B)=0		$WT(B) < TS(T_2)$ T_2 đọc được B
3	Write(A)		RT(A)=100 WT(A)=100			$RT(A) < TS(T_1)$ $WT(A) = TS(T_1)$ — T_1 ghi lên A được
4		Write(B)		RT(B)=200 WT(B)=200		$RT(B) < TS(T_2)$ $WT(B) = TS(T_2)$ — T_2 ghi lên B được
5		Read(C)			RT(C)=200 WT(C)=0	$WT(B) < TS(T_2)$ T_2 đọc được C
6	Read(C)				RT(C)=200 WT(C)=0	$WT(B) < TS(T_1)$ T_1 đọc được C
7	Write(C) ↓ Abort					$RT(B) < TS(T_1)$ T_1 không ghi lên C được

Ví dụ (tt)

T_1 TS=200	T_2 TS=150	T_3 TS=175	A RT=0 WT=0	B RT=0 WT=0	C RT=0 WT=0
Read(B)				RT=200 WT=0	
	Read(A)		RT=150 WT=0		
		Read(C)			RT=175 WT=0
Write(B)				RT=200 WT=200	
Write(A)			RT=200 WT=200		
	Write(C)				
		Write(A)			

Rollback

Giá trị của A đã sao lưu bởi T1
→ T3 không bị rollback và không cần ghi A

Ví dụ (tt)

T_1 TS=150	T_2 TS=200	T_3 TS=175	T_4 TS=255	A RT=0 WT=0
Read(A)				RT=150 WT=0
Write(A)				RT=150 WT=150
	Read(A)			RT=200 WT=0
	Write(A)			RT=200 WT=200
		Read(A)		
			Read(A)	RT=255 WT=200

Rollback

Nhãn thời gian riêng phần (tt)

- Nhận xét
  - Thao tác $read_3(A)$ làm cho giao tác T_3 bị hủy
  - T_3 đọc giá trị của A sẽ được ghi đè trong tương lai bởi T_2
  - Giả sử nếu T_3 đọc được giá trị của A do T_1 ghi thì sẽ không bị hủy

Nhãn thời gian nhiều phiên bản

- Ý tưởng
  - Cho phép thao tác $read_3(A)$ thực hiện
- Bên cạnh việc lưu trữ giá trị hiện hành của A, ta giữ lại các giá trị được sao lưu trước kia của A (phiên bản của A)
- Giao tác T sẽ đọc được giá trị của A ở 1 phiên bản thích hợp nào đó

Nhãn thời gian nhiều phiên bản (tt)

- Mỗi phiên bản của 1 đơn vị dữ liệu X có
  - RT(X)
    - Ghi nhận lại giao tác sau cùng đọc X thành công
  - WT(X)
    - Ghi nhận lại giao tác sau cùng ghi X thành công
- Khi giao tác T phát ra yêu cầu thao tác lên X
  - Tìm 1 phiên bản thích hợp của X
  - Đảm bảo tính khả tuần tự
- Một phiên bản mới của X sẽ được tạo khi hành động ghi X thành công

Nhãn thời gian nhiều phiên bản (tt)

Read(T, X)

```
i="số thứ tự phiên bản sau cùng nhất của A"
While WT(Xi) > TS(T)
      i:=i-1;//lùi lại
Read(Xi);
RT(Xi):= max(RT(Xi), TS(T));
```

Write(T, X)

```
i="số thứ tự phiên bản sau cùng nhất của A"
While WT(Xi) > TS(T)
      i:=i-1;//lùi lại
If RT(Xi) > TS(T)
      Rollback T//Hủy và khởi tạo TS mới
Else
      Tạo phiên bản Ai+1;
      Write(Xi+1);
      RT(Xi+1) = 0;//chưa có ai đọc
      WT(Xi+1) = TS(T);
```

Ví dụ

T_1 TS=150	T_2 TS=200	T_3 TS=175	T_4 TS=255	A_0 RT=0 WT=0	A_1	A_2
Read(A)				RT=150 WT=0		
Write(A)					RT=0 WT=150	
	Read(A)				RT=200 WT=150	
	Write(A)					RT=0 WT=200
		Read(A)			RT=200 WT=150	
			Read(A)			RT=255 WT=200

Ví dụ (tt)

T_1 TS=100	T_2 TS=200	A_0 RT=0 WT=0	B_0 RT=0 WT=0	A_2	B_1
Read(A)		RT=100 WT=0			
	Write(A)		RT=0 WT=200	RT=0 WT=200	
	Write(B)				RT=0 WT=200
Read(B)				RT=100 WT=0	
Write(A)			RT=0 WT=100		

Nhãn thời gian nhiều phiên bản (tt)

- Nhận xét
  - Thao tác đọc
    - Giao tác T chỉ đọc giá trị của phiên bản do T hay những giao tác trước T cập nhật
    - T không đọc giá trị của các phiên bản do các giao tác sau T cập nhật

    → Thao tác đọc không bị rollback
  - Thao tác ghi
    - Thực hiện được bằng cách chèn thêm phiên bản mới
    - Không thực hiện được thì rollback
  - Tốn nhiều chi phí tìm kiếm, tốn bộ nhớ
  - Nên giải phóng các phiên bản quá cũ không còn được các giao tác sử dụng

Nội dung chi tiết

- Các vấn đề truy xuất đồng thời
- Kỹ thuật khóa (locking)
- Kỹ thuật nhãn thời gian (timestamps)
- Kỹ thuật xác nhận hợp lệ (validation)

Giới thiệu

- Ý tưởng
  - Cho phép các giao tác truy xuất dữ liệu 1 cách tự do
  - Kiểm tra tính khả tuần tự của các giao tác
    - Trước khi ghi, tập hợp các đơn vị dữ liệu của 1 giao tác sẽ được so sánh với tập đơn vị dữ liệu của những giao tác khác
    - Nếu không hợp lệ, giao tác phải rollback
- Trong khi nhãn thời gian
  - Lưu giữ lại các phiên bản của đơn vị dữ liệu
- Thì xác nhận tính hợp lệ
  - Quan tâm đến các giao tác đang làm gì

Xác nhận hợp lệ

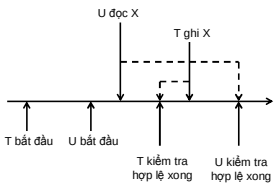
- Một giao tác có 3 giai đoạn
  - (1) Đọc - Read set - RS(T)
    - Đọc tất cả các đơn vị dữ liệu có trong giao tác
    - Tính toán rồi lưu trữ vào bộ nhớ phụ
    - Không sử dụng cơ chế khóa
  - (2) Kiểm tra hợp lệ - Validate
    - Kiểm tra tính khả tuần tự
  - (3) Ghi - Write set - WS(T)
    - Nếu (2) hợp lệ thì ghi xuống CSDL
- Chiến lược cơ bản
  - Nếu T_1, T_2, …, T_n là thứ tự hợp lệ thì kết quả sẽ conflict-serializable với lịch tuần tự $\{T_1, T_2, \ldots, T_n\}$

Xác nhận hợp lệ (tt)

- Bộ lập lịch xem xét 3 tập hợp
  - START
    - Tập các giao tác đã bắt đầu nhưng chưa kiểm tra hợp lệ xong
    - START(T) ghi nhận thời điểm bắt đầu của T
  - VAL
    - Tập các giao tác được kiểm tra hợp lệ nhưng chưa hoàn tất ghi
      - Các giao tác đã hoàn tất giai đoạn 2
    - VAL(T) ghi nhận thời điểm T kiểm tra xong
  - FIN
    - Tập các giao tác đã hoàn tất việc ghi
      - Các giao tác đã hoàn tất giai đoạn 3
    - FIN(T) ghi nhận thời điểm T hoàn tất

Xác nhận hợp lệ (tt)

- Vấn đề 1



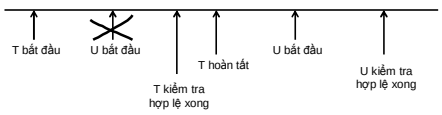
- T đã kiểm tra hợp lệ xong
- T chưa hoàn tất ghi thì U bắt đầu đọc
- RS(U) ∩ WS(T) = {X}

- U có thể không đọc được giá trị X ghi bởi T

→ Rollback U

Xác nhận hợp lệ (tt)

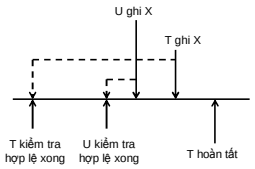
- Vấn đề 1



Sau khi T hoàn tất thì U mới bắt đầu

Xác nhận hợp lệ (tt)

- Vấn đề 2



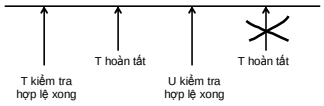
- T đã kiểm tra hợp lệ xong
- T chưa hoàn tất ghi thì U kiểm tra hợp lệ
- WS(U) ∩ WS(T) = {X}

- U có thể ghi X trước T

→ Rollback U

Xác nhận hợp lệ (tt)

- Vấn đề 2

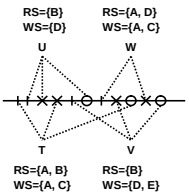


Sau khi T hoàn tất thì U mới được kiểm tra hợp lệ

Xác nhận hợp lệ (tt)

- Qui tắc
  - (1) - Nếu có T chưa hoàn tất mà U bắt đầu
    - Kiểm tra RS(U) ∩ WS(T) = ∅
  - (2) - Nếu có T chưa hoàn tất mà U kiểm tra hợp lệ
    - Kiểm tra WS(U) ∩ WS(T) = ∅

Ví dụ



I = bắt đầu
X = kiểm tra hợp lệ
O = hoàn tất

Khi U kiểm tra hợp lệ:

Khi T kiểm tra hợp lệ:

U đã kiểm tra hợp lệ xong nhưng chưa hoàn tất nên kiểm tra WS(U) và [RS(T), WS(T)]

Khi V kiểm tra hợp lệ:

Vì V bắt đầu trước khi U hoàn tất nên kiểm tra RS(V) và WS(U)

Khi W kiểm tra hợp lệ:

U hoàn tất trước khi W bắt đầu → kg kiểm tra

Vì W bắt đầu trước khi T hoàn tất nên kiểm tra RS(W) và WS(T) → A

V kiểm tra hợp lệ xong nhưng chưa hoàn tất nên kiểm tra WS(V) và [RS(W), WS(W)] → D

→ W không hợp lệ và phải rollback

Nhận xét

- Kỹ thuật nào hiệu quả hơn???
  - Khóa (locking)
  - Nhãn thời gian (timestamps)
  - Xác nhận hợp lệ (validation)
- Dựa vào
  - Lưu trữ
    - Tỷ lệ với số lượng đơn vị dữ liệu
  - Khả năng thực hiện
    - Các giao tác ảnh hưởng với nhau như thế nào? Nhiều hay ít?

Nhận xét (tt)

	Khóa	Nhãn thời gian	Xác nhận hợp lệ
Delay	Trì hoãn các giao tác, ít rollback	Không trì hoãn các giao tác, nhưng gây ra rollback	
Rollback		Xử lý rollback nhanh	Xử lý rollback chậm
Storage	Phụ thuộc vào số lượng đơn vị dữ liệu bị khóa	Phụ thuộc vào nhãn đọc và ghi của từng đơn vị dữ liệu	Phụ thuộc vào nhãn WS và RS của các giao tác hiện hành và 1 vài giao tác đã hoàn tất sau 1 giao tác bắt đầu nào đó
		Sử dụng nhiều bộ nhớ hơn	

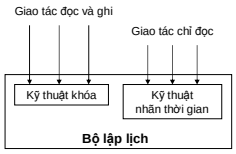
ảnh hưởng nhiều

ảnh hưởng ít

Nhận xét (tt)

- Khóa & nhãn thời gian
  - Nếu các giao tác chỉ thực hiện đọc không thôi thì kỹ thuật nhãn thời gian tốt hơn
    - Ít có tình huống các giao tác cố gắng đọc và ghi cùng 1 đơn vị dữ liệu
  - Nhưng kỹ thuật khóa sẽ tốt hơn trong những tình huống xãy ra tranh chấp
    - Kỹ thuật khóa thường hay trì hoãn các giao tác để chờ xin được khóa
      - Dẫn đến deadlock
    - Nếu có các giao tác đọc và ghi cùng 1 đơn vị dữ liệu thì việc rollback là thường xuyên hơn

Nhận xét (tt)



Kết luận

Mỗi kỹ thuật đều có ưu việt riêng