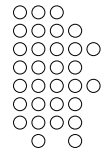
Chương 2

Những vấn đề khác trong điều kiển đồng thời



Nội dung chi tiết

- Quay lui dây chuyền (cascading rollback)
- Lịch khả phục hồi (recoverable schedule)
- Deadlock
  - Phát hiện (detection)
  - Ngăn ngừa (prevention)

Ví dụ

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

Ví dụ (tt)

S

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

Tính nhất quán bị vi phạm
→ T_2 cũng phải rollback

Ví dụ (tt)

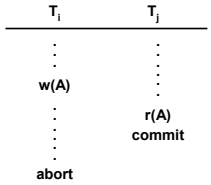
S

T_1 200	T_2 150	T_3 175	A RT=0 WT=0	B RT=0 WT=0	C RT=0 WT=0	
	Write(B)			RT=0 WT=150		
Read(B)				RT=200 WT=150		
	Read(A)		RT=150 WT=0			
		Read(C)			RT=175 WT=0	
	Write(C) **Abort**			**RT=0 WT=0**		**→ Phục hồi giá trị của B**
		Write(A)	RT=150 WT=175			

Quay lui dây chuyền

T_1	T_2	T_3	T_4
.	.	.	.
.	.	.	.
w(A)	.	.	
.	**r(A)**	.	
.	.	.	
.	.	**r(A)**	
abort	.	.	
	abort	.	
		abort	

Lịch khả phục hồi



- Xét mỗi cặp T_i và T_j sao cho
  - T_j đọc dữ liệu sau khi T_i ghi
  - T_i phải được hoàn tất (commit) trước khi T_j hoàn tất
    - Ký hiệu c_i: giao tác thứ i hoàn tất

Lịch khả phục hồi (tt)

S_1 : $w_1(A); w_1(B); w_2(A); r_2(B); c_1; c_2;$

T_2 đọc B sau T_1 ghi B

T_1 hoàn tất trước T_2

S_1 tuần tự và khả phục hồi

Lịch khả phục hồi (tt)

S_2 : $w_2(A); w_1(B); w_1(A); r_2(B); c_1; c_2;$

T_1 ghi B trước T_2 đọc B sau

T_1 hoàn tất trước T_2

S_2 không khả tuần tự nhưng khả phục hồi

Lịch khả phục hồi (tt)

S_3 : $w_1(A); w_1(B); w_2(A); r_2(B); c_2; c_1;$

T_1 ghi B trước T_2 đọc B sau

T_2 hoàn tất trước T_1

S_3 khả tuần tự nhưng không khả phục hồi

Lịch khả phục hồi (tt)

- Nhận xét
  - Muốn khôi phục đôi khi cần quay lui dây chuyền
  - Nhưng quay lui dây chuyền không thể xãy ra
    - Tốn nhiều chi phí

$\rightarrow$ Lịch không quay lui dây chuyền (cascadeless schedule)

Lịch không quay lui dây chuyền

T_i	T_j
.	.
.	.
w(A)	.
commit	.
	.
	.
	r(A)

Các giao tác chỉ đọc những giá trị đã được hoàn tất

Lịch không quay lui dây chuyền (tt)

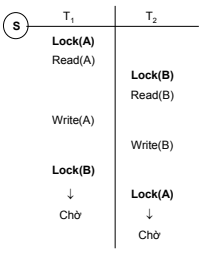
- Khả phục hồi
  S_1 : $w_1(A); w_1(B); w_2(A); r_2(B); c_1; c_2;$
- Ngăn ngừa quay lui dây chuyền
  S_1 : $w_1(A); w_1(B); w_2(A); c_1; r_2(B); c_2;$
- $\rightarrow$ Các lịch ngăn ngừa quay lui dây chuyền đều khả phục hồi

Nội dung chi tiết

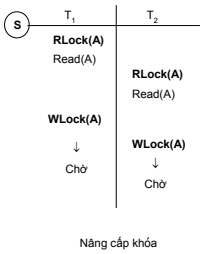
- Quay lui dây chuyền (cascading rollback)
- Lịch khả phục hồi (recoverable schedule)
- Deadlock
  - Phát hiện (detection)
  - Ngăn ngừa (prevention)

Deadlock

- Nhắc lại 2 tình huống



Qui tắc khóa 2PL



Nâng cấp khóa

Deadlock (tt)

- Hệ thống rơi vào trạng thái deadlock khi
  - Các giao tác phải chờ đợi lẫn nhau để được thao tác lên các đơn vị dữ liệu bị khóa bởi chúng
  - Và không một giao tác nào có thể thực hiện tiếp công việc của mình

Giải quyết Deadlock

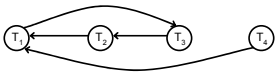
- Phát hiện
  - Cho phép trạng thái deadlock xãy ra và sau đó cố gắng khôi phục lại hệ thống
    - Chọn 1 giao tác để rollback
  - Phương pháp
    - Đồ thị chờ (wait-for graph)
- Ngăn ngừa
  - Quản lý các giao tác sao cho không bao giờ có deadlock
  - Phương pháp
    - Sắp thứ tự tài nguyên (resource ordering)
    - Timeout
    - Wait-die
    - Wound-wait

Đồ thị chờ

- Đồ thị gồm
  - Đỉnh là các giao tác đang giữ khóa hoặc đang chờ khóa
  - Cung đi từ đỉnh T sang U khi
    - U đang giữ khóa trên đơn vị dữ liệu A
    - T đang chờ khóa trên A
    - T không thể khóa đơn vị dữ liệu A nếu U không giải phóng khóa
- Nếu đồ thị chờ không có chu trình
  - Các giao tác có thể hoàn tất
- Ngược lại
  - Không một giao tác nào trong chu trình có thể tiếp tục thực hiện $\rightarrow$ deadlock

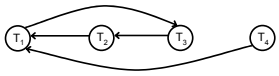
Ví dụ

	T_1	T_2	T_3	T_4
1	L(A); R(A)			
2		L(C); R(C)		
3			L(B); R(B)	
4				L(D); R(D)
5		L(A)		
6		↓	L(C)	
7		Chờ	↓	L(A)
8	L(B)		Chờ	↓
	↓			Chờ
	Chờ			



Ví dụ (tt)

	T_1	T_2	T_3	T_4
1	L(A); R(A)			
2		L(C); R(C)		
3			L(B); R(B)	
4				L(D); R(D)
5		L(A)		
6			L(C)	
7				L(A)
8	L(B)			



Sắp thứ tự tài nguyên

- Áp đặt một thứ tự nào đó lên các đơn vị dữ liệu
- Nếu các giao tác thực hiện khóa những đơn vị dữ liệu theo thứ tự này
- Thì không có deadlock xãy ra trong khi chờ đợi

- Chứng minh
  - Bài tập về nhà

Ví dụ

- Giả sử các đơn vị dữ liệu được sắp thứ tự theo alphabet

  T_1 : l(A); r(A); l(B); w(B); u(A); u(B);

  T_2 : l(C); r(C); l(A); w(A); u(C); u(A);

  T_3 : l(B); r(B); l(C); w(C); u(B); u(C);

  T_4 : l(D); r(D); l(A); w(A); u(D); u(A);

Ví dụ (tt)

T_1 : l(A); r(A); l(B); w(B); u(A); u(B);

T_2 : l(A); l(C); r(C); w(A); u(C); u(A);

T_3 : l(B); r(B); l(C); w(C); u(B); u(C);

T_4 : l(A); l(D); r(D); w(A); u(D); u(A);

Ví dụ (tt)

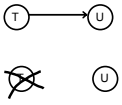
	T_1	T_2	T_3	T_4
1	L(A); R(A)			
2		L(A)		
3		↓	L(B); R(B)	
4		Chờ		L(A)
5			L(C); W(C)	↓
6			U(B); U(C)	Chờ
7	L(B); R(B)			
8	U(A); U(B)			
9		L(A); L(C)		
10		R(A); W(C)		
11		U(C); U(A)		
12				L(A); L(D)
13				R(D); W(A)
14				U(D); U(A)

Timeout

- Giới hạn các giao tác chỉ được thực hiện trong 1 khoảng thời gian nào đó
- Nếu giao tác vượt quá thời gian này
- Thì giao tác phải bị rollback

Wait-die

- Mỗi giao tác sẽ được gán một nhãn ghi nhận thứ tự xuất hiện, kí hiệu: ts(T)
- Xét 2 giao tác T và U
  - U đang giữ khóa trên đơn vị dữ liệu A
  - T muốn khóa đơn vị dữ liệu A
  - T sẽ chờ-wait U khi ts(T) < ts(U)
  - Ngược lại T sẽ bị hủy-die và bắt đầu làm lại ở 1 thời điểm khác



Ví dụ

	T_1	T_2	T_3	T_4
1	L(A); R(A)			
2		L(A)		
3		↓	L(B); R(B)	
4		**Dies**		L(A)
5			L(C); W(C)	↓
6			U(B); U(C)	**Dies**
7	L(B); R(B)			
8	U(A); U(B)			

Ví dụ (tt)

- T_2 bắt đầu trước T_4

	T_1	T_2	T_3	T_4
1	L(A); R(A)			
2		L(A)		
3		↓	L(B); R(B)	
4		**Dies**		L(A)
5			L(C); W(C)	↓
6			U(B); U(C)	**Dies**
7	L(B); R(B)			
8	U(A); U(B)			
9		L(A); L(C)		
10				L(A)
11		R(A); W(C)		↓
12		U(C); U(A)		**Dies**
13				L(A); L(D)
14				R(D); W(A)
15				U(D); U(A)

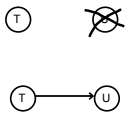
Ví dụ (tt)

- T_4 bắt đầu trước T_2

	T_1	T_2	T_3	T_4
1	L(A); R(A)			
2		L(A)		
3		↓	L(B); R(B)	
4		**Dies**		L(A)
5			L(C); W(C)	↓
6			U(B); U(C)	**Dies**
7	L(B); R(B)			
8	U(A); U(B)			
9				L(A); L(D)
10		L(A)		
11		↓		R(D); W(A)
12		**Waits**		U(D); U(A)
13		L(A); L(C)		
14		R(A); W(C)		
15		U(C); U(A)		

Wound-wait

- Mỗi giao tác sẽ được gán một nhãn ghi nhận thứ tự xuất hiện, kí hiệu: ts(T)
- Xét 2 giao tác T và U
  - U đang giữ khóa trên đơn vị dữ liệu A
  - T muốn khóa đơn vị dữ liệu A
  - T buộc U rollback và trao khóa lại cho T-wound khi ts(T) < ts(U)
    - Ngoại lệ: nếu U đã kết thúc và giải phóng khóa, U sẽ không rollback
  - Ngược lại T sẽ chờ-wait U



Ví dụ

	T_1	T_2	T_3	T_4
1	L(A); R(A)			
2		L(A)		
3		↓	L(B); R(B)	
4		**Waits**		L(A)
5	L(B); R(B)		↓	↓
6	U(A); U(B)		**Wound**	**Waits**
7		L(A); L(C)		
8		R(A); W(C)		
9		U(C); U(A)		
10				L(A); L(D)
11				R(D); W(A)
12				U(D); U(A)
13			L(B); R(B)	
14			L(C); W(C)	
15			U(B); U(C)	

Nhận xét

- Timeout
  - Đơn giản
  - Khó chọn được khoảng thời gian timeout thích hợp
  - Có hiện tượng starvation
    - Giao tác lập đi lập lại quá trình: bắt đầu, deadlock, rollback
- Resource ordering
  - Không thực tế
  - Chờ đợi nhiều → tiềm ẩn của deadlock

Nhận xét (tt)

- Wait-die và Wound-wait
  - Không có starvation
  - Wound-wait ít rollback các giao tác hơn wait-die
  - Dễ cài đặt hơn wait-for graph
  - Có thể rollback những giao tác không gây ra deadlock
- Wait-for graph
  - Nếu đồ thị quá lớn sẽ tốn nhiều thời gian phân tích
  - Rất phức tạp khi CSDL phân tán
  - Giảm tối thiểu rollback các giao tác
    - Chỉ rollback 1 trong những giao tác gây ra deadlock