SERIALIZABILITY AND RECOVERABILITY

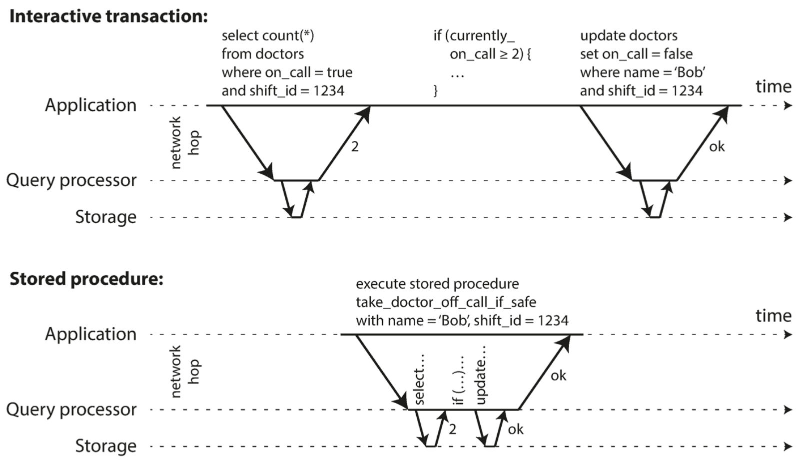
26

Serial Execution

- Run applications one at a time
  - long-running data analytic queries
  - transaction in stored procedure
- Assumptions
  - No I/O (DB entirely in RAM)
  - Single CPU
  - No interactivity
- Ex: Redis

27

Serial Execution



28

28

Concurrent Transactions

- Transactions may execute in parallel
  - processor utilization
  - improved average response time
- Schedules
  - Interleaving of update operations of transactions
  - Updates for transaction X must occur in schedule in the same order as in X
- Serial Schedule
  - Ensures isolation of transactions
  - Ensures consistency if each transaction preserves consistency

29

29

Serial Schedule

- T_1

  read A
  A := A - 50
  write A
  read B
  B := B + 50
  write B

- T_2

  read A
  temp := A / 10
  A := A - temp
  write A
  read B
  B := B + temp
  write B

30

30

Conflicting Operations

Read-Write Conflict

- Suppose initial value of A is $100
- Schedule 1:
```
Read A
Write 150 to A
```
- Schedule 2:
```
Write 150 to A
Read A
```

Write-Write Conflict

- Suppose initial value of A is $100
- Schedule 1:
```
Write 150 to A
Write 200 to A
```
- Schedule 2:
```
Write 200 to A
Write 150 to A
```

31

31

Conflict Serializability

- Operations O1 and O2 conflict if:
  - O1 ≡ read A, O2 ≡ write A
  - O1 ≡ write A, O2 ≡ read A
  - O1 ≡ write A, O2 ≡ write A
- Schedules S and S' are conflict equivalent if S' can be obtained from S by swapping non-conflicting operations
- S is serializable if S is equivalent to a serial schedule

32

32

Non-Serial but Serializable Schedule

T_1	T_2
read A A := A - 50 write A	
	read A temp := A / 10 A := A - temp write A
read B B := B + 50 write B	
	read B B := B + temp write B

33

33

Non-Serial but Serializable Schedule

- T_1

  read A
  A := A - 50
  write A

  read B
  B := B + 50
  write B

- T_2

  read A
  temp := A / 10
  A := A - temp

  write A
  read B
  B := B + temp
  write B

34

34

Non-Serial but Serializable Schedule

- T_1

  read A
  A := A - 50
  write A

  read B
  B := B + 50
  write B

- T_2

  read A

  temp := A / 10
  A := A - temp
  write A
  read B
  B := B + temp
  write B

35

35

Non-Serial but Serializable Schedule

- T_1
  read A
  A := A - 50
  write A
  read B
  B := B + 50
  write B

- T_2
  read A
  temp := A / 10
  A := A - temp
  write A
  read B
  B := B + temp
  write B

36

36

Non-Serializable Schedule

- T_1
  read A
  A := A - 50

- T_2
  read A
  temp := A / 10
  A := A - temp
  write A
  read B

- T_1
  write A
  read B
  B := B + 50
  write B

- T_2
  B := B + temp
  write B

37

37

Non-Serializable Schedule

T_1	T_2
read A	
A := A - 50	
	read A
	temp := A / 10
	A := A - temp
	write A
write A	
read B	
B := B + 50	
	read B
write B	
	B := B + temp
	write B

38

38

Recoverability

- Successful transaction ends with commit operation
- A schedule is **recoverable** if, whenever T_i reads data item written by T_j, commit of T_i follows commit of T_j

39

39

Recoverability

- The following schedule is not recoverable:

T_1	T_2
write A	
	read A
	commit
read B	
commit	

40

40

Cascading Rollbacks

- The following schedule is recoverable:

T_1	T_2	T_3
read A		
write A		
	read A	
	write A	
		read A
commit		
	commit	
		commit

41

41

Cascading Rollbacks

- The following schedule is cascadeless:

T_1	T_2	T_3
read A		
write A		
commit		
	read A	
	write A	
	commit	
		read A
		commit

42

42

Questions

- How could you use locks to ensure serializability?
- How could you use locks to ensure recoverability? Cascadeless schedules?

43

43

CONCURRENCY CONTROL

44

Implementing Isolation

- Schedules must be
  - serializable
  - recoverable
  - preferably cascadeless
- One-at-a-time execution inefficient
  - Transactions can use the CPU(s) while other transactions are blocked waiting for I/O operations to complete
- Concurrency control

45

Scheduling

- Two operations conflict if they operate on the same data item and at least one of them is a write
  - read-write conflict
  - write-write conflict
- Two read operations can never conflict
- Concurrency control schemes are classified by how they synchronize read and write operations (locking, ordering via timestamps, …)

46

46

Two Scheduling Approaches

- Pessimistic - if something can go wrong, it will
  - Operations explicitly synchronized during execution
- Optimistic - in general, nothing will go wrong
  - Synchronization happens at the end of the transaction

47

47

Lost Updates

- T_1
  read A
  A := A - 50

  write A

- T_2

  read A
  A := A + 100

  write A

48

48

Locking for Isolation

- T_1
  lock-X A
  read A
  A := A - 50
  write A
  unlock A

- T_2

  lock-X A
  read A
  A := A + 100
  write A
  unlock A

49

49

Examples of lock coverage

- We could have one lock per object
- … or one lock for the whole database
- … or one lock for a category of objects
  - Tree
  - Table, row, column
- All transactions must use the same rules!
- “Write” locks for updates

50

50

Lock-Based Concurrency Control

- Two modes of locking:
  - shared (read-only)
  - exclusive (read-write)
- Lock acquisition
  - transaction blocks if lock not available
  - update shared lock to exclusive
  - downgrade exclusive lock to shared
- Possibility of **deadlock**

51

51

Lock-Based Concurrency Control

- Lock Compatibility

Request by T_i \ Held by T_j	S	X
S	true	false
X	false	false

- Lock manager implemented using readers-writers algorithm

52

52

Locking for Isolation

- T_1

lock-X A; read A
A := A - 50
write A; unlock A

lock-X B; read B
B := B + 50
write B; unlock B

- T_2

lock-X B; read B
B := B - 100
write B; unlock B

lock-X A; read A
A := A + 100
write A; unlock A

53

53

Locking for Serializability

- T_1
  lock-X A; lock-X B
  read A; A := A - 50
  write A
  read B; B := B + 50
  write B
  unlock A; unlock B

- T_2
  lock-X B; lock-X A
  read B; B := B - 100
  write B
  read A; A := A + 100
  write A
  unlock B; unlock A

54

54

Locking for Serializability

- T_1
  lock-X A; read A
  A := A - 50
  write A;
  lock-X B; read B
  B := B + 50
  write B;
  unlock A; unlock B

- T_2
  lock-X B; read B
  B := B - 100
  write B;
  Lock-X A; read A
  A := A + 100
  write A;
  unlock B; unlock A

55

55

Two-Phase Locking Protocol (2PL)

- Protocol:
  - Phase 1 (Growing)
    - transaction may obtain or upgrade locks
    - transaction may not release or downgrade locks
  - Phase 2 (Shrinking)
    - transaction may release or downgrade locks
    - transaction may not obtain or upgrade locks
- 2PL ensures serializability
  - Prevent cycles in temporal dependencies
- 2PL increases chance of deadlock

56

56

Locking for Recoverability

- T_1

  lock-X A; read A
  A := A – 50; write A
  lock-X B; read B
  B := B + 50; write B
  unlock A; unlock B

  commit

- T_2

  lock-X B; read B
  B := B – 100; write B
  lock-X A; read A
  A := A + 50; write A
  unlock B; unlock A

  commit

57

57

Locking for Recoverability

- T1

```
lock-X A
read A;  A := A - 50
write A;
lock-X B
read B;  B := B + 50
write B
commit; unlock A; unlock B
```

- T2

```
lock-X B
read B;  B := B - 100
write B
lock-X A
read A;  A := A + 50
write A
commit; unlock B; unlock A
```

58

58

Variations on 2PL

- Strict 2PL
  - Transaction must hold all **exclusive** locks until it commits or aborts
  - Ensures cascadeless schedules
- Rigorous 2PL
  - Transaction must hold **all** locks until it commits or aborts
  - Ensures transactions can be serialized in the order in which they commit

59

59