CLOUD COMPUTING CONCEPTS

with Indranil Gupta (Indy)

CONCURRENCY CONTROL

Lecture C

SERIAL EQUIVALENCE

Concurrent Transactions

- To prevent transactions from affecting each other
  - Could execute them one at a time at server
  - But reduces number of concurrent transactions
  - *Transactions per second* directly related to revenue of companies
    - This metric needs to be maximized
- Goal: increase concurrency while maintaining correctness (ACID)

Serial Equivalence

- An interleaving (say O) of transaction operations is serially equivalent iff (if and only if):
  - There is some ordering (O’) of those transactions, one at a time, which
  - Gives the same end-result (for all objects and transactions) as the original interleaving O
  - Where the operations of each transaction occur consecutively (in a batch)
- Says: Cannot distinguish end-result of real operation O from (fake) serial transaction order O’

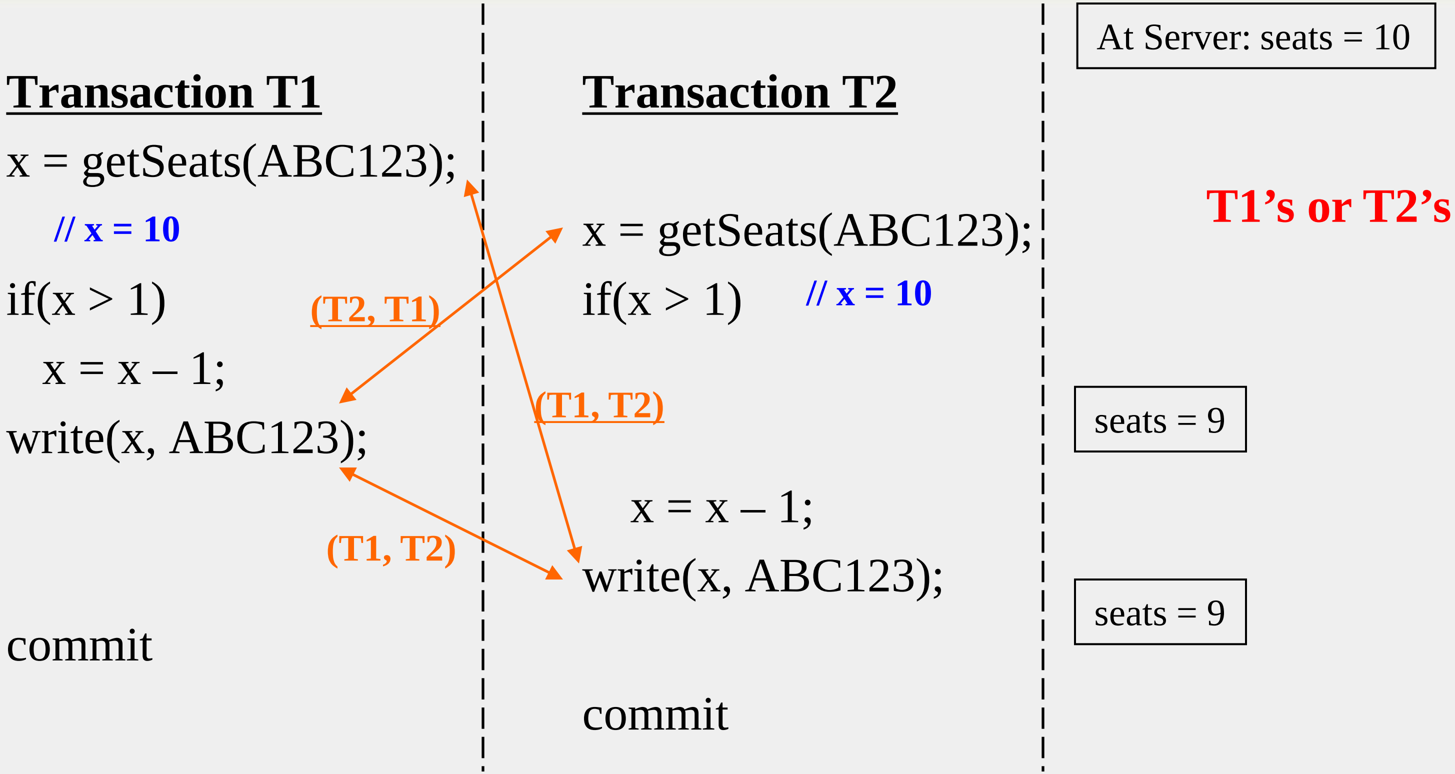
Checking for Serial Equivalence

- An operation has an effect on
  - The server object if it is a write
  - The client (returned value) if it is a read
- Two operations are said to be conflicting operations, if their *combined effect* depends on the order they are executed
  - read(x) and write(x)
  - write(x) and read(x)
  - write(x) and write(x)
  - NOT read(x) and read(x): swapping them doesn’t change end-results
  - NOT read/write(x) and read/write(y): swapping them ok

Checking for Serial Equivalence (2)

- *Two transactions are serially equivalent if and only if all pairs of conflicting operations (pair containing one operation from each transaction) are executed in the same order (transaction order) for all objects (data) they both access.*
  - Take all pairs of conflict operations, one from T1 and one from T2
  - If the T1 operation was reflected first on the server, mark the pair as "(T1, T2)," otherwise mark it as "(T2, T1)"
  - All pairs should be marked as either "(T1, T2)" or all pairs should be marked as "(T2, T1)."

1. Lost Update Problem – Caught!



Transaction T1

```
x = getSeats(ABC123);
    // x = 10
if(x > 1)
   x = x – 1;
write(x, ABC123);

commit
```

Transaction T2

```
x = getSeats(ABC123);
if(x > 1)      // x = 10

    x = x – 1;
write(x, ABC123);

commit
```

At Server: seats = 10

seats = 9

seats = 9

T1’s or T2’s update was lost!

2. Inconsistent Retrieval Problem – Caught!

At Server:
ABC123 = 10
ABC789 = 15

Transaction T1		**Transaction T2**
x = getSeats(ABC123);		
y = getSeats(ABC789);		
write(x-5, ABC123);	(T1, T2)	
		x = getSeats(ABC123);
		y = getSeats(ABC789);
write(y+5, ABC789);	(T2, T1)	// x = 5, y = 15
		print(“Total:” x+y);
commit		// Prints “Total: 20”
		commit

T2’s sum is the wrong value!
Should have been “Total: 25”

What's Our Response?

- At commit point of a transaction T, check for serial equivalence with all other transactions
  - Can limit to transactions that overlapped with T
- If not serially equivalent
  - Abort T
  - Roll back (undo) any writes that T did to server objects

Can We Do Better?

- Aborting => wasted work
- Can you prevent violations from occurring?