Slide 1

Today's Topics - Transactions Chapter 12.

Today we will look at Transactions on single servers.

- What are Transactions?
- Concurrency Control
- Recoverability
- Nested Transactions
- Locks 12.4
- Introduction to Optimistic Concurrency Control
- Introduction to Timestamps.

Reading all of Chapter 12.

Slide 2

What are Transactions?

Suppose that you are booking a flight from Sundsvall to Florida.

You go to the travel agents they tell you that you must.

- Fly Sundsvall to Stockholm
- Stockholm to Chicago
- Chicago to Florida.

You say OK, book the flights they start book the flights one by one and discover on the last flight that it is full. The next free flight from Chicago to Florida is the next day. So you have to wait all a whole day at Chicago.

Slide 3

What are Transactions?

- The situation on the previous is not optimal.
- You want to either book all the flights or book none of them. You don't want a half completed booking.
- Especially if the half completed booking commits you to something that you don't want (spending a night at Chicago).
- A transaction is a sequence of operations with an all or nothing behavior.
- The problems comes is that other people might be booking flights at the same time and hence some flights might get full while you think they are empty.

Slide 4

What are Transactions?

Transactions are atomic in the sense that:

1. they are free from interference by operations being performed on behalf of other concurrent client;
2. either all of the operations must be completed successfully or they must have no effect at all (even in the presence of server crashes)

Slide 5

ACID

Transactions should satisfy the mnemonic 'ACID':

- **A**tomicity: a transaction must be all or nothing;
- **C**onsistnecy: a transaction takes the same from one consistent state to another consistent state;
- **I**solation; transactions should be run as if they are they only transaction running (they should not be interfered with by another transaction);
- **D**urability: when the transaction commits it should be recoverable in case of server failure (if the server crashes after I have booked my flight, the flight is still booked).

Slide 6

Problems with Transactions - Concurrency Control

- The problem with transaction is that we want to do them efficiently.
- An inefficient way of doing things would be to stop access to the server by anybody else while when a transaction has started and only grant access when this person has finished.
- Instead we allow many transactions to go on at the same time and try to stop bad things happening when we do things at the same time.

Problems with Transactions - Concurrency Control

Slide 7 Two things can go wrong when we merge transactions.

- The Lost Update Problem
- Inconsistent retrievals

To illustrate what is going on we look at a bank account example.

The Lost update problem

Consider two transaction T and U where $A = \$100$, $B = \$200$ and

$C = \$300$.

Slide 8

Transaction T:	**Transaction U:**
`balance = b.getbalance();`	`balance = b.getBalance();`
`b.setBalance(balance*1.1);`	`b.setBalance(balance*1.1);`
`a.withdraw(balance/10);`	`c.withdraw(balance/10);`

Executing T and U should result in b being increased by 10% and 10% again so b should have the value \$242 dollars afterwards.

Slide 9

The lost update problem

Now consider the following interleaving of the operations :

`bal = b.getBalance();`	\$200		
		`bal = b.getBalance();`	\$220
		`b.setBalance(bal*1.1);;`	\$200
`b.setBalance(bal*1.1);`	\$220		
`a.withdraw(bal/10);`	\$80		
		`c.withdraw(bal/10);`	\$280

We see that we lost one of the updates to b.

Slide 10

Inconsistent retrievals

Two transaction V and W.

Transaction V:	**Transaction W:**
`a.withdraw(100);` `b.deposit(100);`	`calculate Branch total.`

The inconsistent retrievals Problem

Slide 11

Consider the following interleaving:

`a.withdraw(100);` $100	
	`total = a.getBalance();` $100
	`total += b.getBalance();` $300
`b.deposit(100)` $300	

The problem is that the sum will be the wrong value.

Serial Equivalence

Slide 12

- If we have a set of transactions and we don't have any particular order on them then we could say a correct result is some sequence of them.
- For example consider the set of transaction S, T, U then we could take any order: $S; T; U$,$S; U; T$,$T; S; U$,$T; U; S$,$U; S; T$,$U; T; S$.
- We say an interleaving of the operations of a transaction is `Serially equivalence` if the result if equivalent to some sequence of transactions.
- The goal of concurrency control is to ensure serial equivalence while trying to be as efficient as possible.

Slide 13

Serial Equivalence

A serially equivalent interleaving of V and W.

Transaction V:	**Transaction W:**
`a.withdraw(100);` `b.deposit(100);`	`calculate Branch total.`

`a.withdraw(100);`	\$100		
`b.deposit(100)`	\$300		
		`total = a.getBalance();`	\$100
		`total += b.getBalance();`	\$300

Slide 14

Serial Equivalence

A serially equivalent interleaving of T and U.

`bal = b.getBal()`	\$200		
`b.setBal(bal*1.1)`	\$220		
		`bal = b.getBal();`	\$220
		`b.setBal(bal*1.1);`	\$242
`a.withdraw(bal/10);`	\$80		
		`c.withdraw(bal/10);`	\$278

Conflicting Operations

Slide 15

- When a pair of operations conflicts we mean that their combined effects depends on the order in which they are executed.
- Consider only `Read` and `Write` operations.

Operations	Conflict
`read read`	No
`read write`	Yes
`write write`	Yes

Serial Equivalence

Slide 16

For two transaction to be `Serially Equivalent`, it is necessary and sufficient that all pairs of conflicting operations of the two transactions be executed in the same order at all the data they both access.

Consider two transactions T and U defined as follows:

- T: `x=read(i);y=read(j);write(k,x+y);`
- U: `write(j,10);write(i,20);`

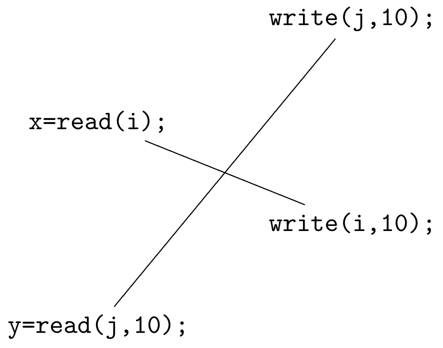
The conflicting pairs of operations are as follows:

- `x=read(i)` and `write(i,20);`
- `y=read(j); write(j,10);`

Slide 17

Conflicting pairs

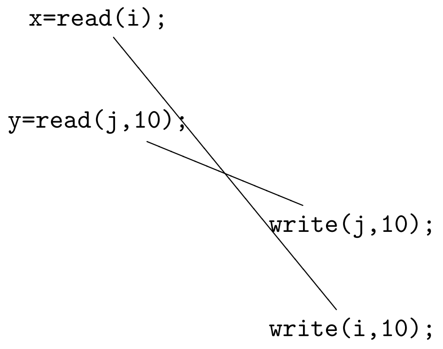
The following interleaving is not serially equivalent.



Slide 18

Serial Equivalence - Conflicting Operations

The following is serially equivalent:



Conflicting Operations - Serial Equivalence

Slide 19

- It might seem that we haven't got so far.
- But, before we had a global definition of serial equivalence.
- Now we have a local condition to satisfy in terms of conflicting pairs.
- There are three commonly used alternatives for concurrency control:
  - Locking (Widely used but can lead to deadlocks);
  - Optimistic concurrency control (assume that nothing bad will happen later repair any mess that has been created)
  - Timestamp ordering.

Locking

Slide 20

- Basic idea: A transaction must be scheduled so that their effect on shared data is serially equivalent.
- A server can achieve serial equivalence of transactions by serializing access to the objects.
- Simple example, the use of exclusive locks. Only one object can read or write at a time. (Can be more refined allow separate locks for reading and writing).
- If you can't lock the data you have to wait.

Slide 21

Exclusive Locks

Operation	Locks	Operations	Locks
`bal = b.getBal();`	Lock B		
`b.setBal(bal*1.1)`			
`a.withdraw(bal/10)`	lock A	`bal = b.getBal()`	wait
`closeTransaction`	unlock A,B		
			lock B
		`b.setBalance(bal*1.1)`	
		`c.withdraw(bal/10)`	lock C
		`closeTransaction`	unlock B,C

Slide 22

Strict Two Phase Locking

- *Two Phase Locking* Transaction is not allowed any new locks after it has released a lock. Two phases growing and shrinking phase. Growing phase all locks are acquired, shrinking phase locks are given back.
- *Strict Two Phase Locking* Any locks acquired are not given back until the transaction completed or aborts (ensures durability). Locks must be held until all the objects it updated have been written to permanent storage.

Deadlocks

Slide 23

- When an item is locked another process might be waiting for it. The process might then be waiting for a lock held by the waiting process.

Insert figure 12.20 , 12.21

Deadlock Prevention

Slide 24

- Simple solution, lock everything the beginning of each transaction. Not terrible efficient.
- Lock things in a predefined order. (Also not so efficient, might result in premature locking).

Deadlock Detection

Slide 25

- Look for cycles in the wait graph.
- The lock manager keeps track of who is waiting for what lock and checks for cycles.
- When a deadlock is detected you have to abort a transaction.

Optimistic Concurrency Control

Slide 26

- Locking is not so efficient.
- Optimistic concurrency control.
  - Proceed with all transaction assuming that things are working well.
  - Abort transaction which didn't complete correctly
  - Works on the assumption that on the whole transaction don't interfere that much.

Timestamp Ordering

Slide 27

- In concurrency control schemes based on timestamp ordering, each operation in a transaction is validated when it is carried out.
- If the operation cannot be validated the transaction is aborted immediately and then can be restarted by the client.
- Each transaction is assigned a unique timestamp value when it starts;
- The timestamp defines its position in the time sequence of transactions.
- Request from transaction can be ordered according to their timestamps.

Timestamp Ordering

Slide 28

The basic timestamp rule is as follows:

- A transaction's request to write an object is valid only if that object was last read and written by earlier transactions. A transaction's request to read an object is valid only if that object was last written by an earlier transactions.