CS186 Section
Week 9

Transaction and Concurrency Intro

Today

1. Transactions (ACID)
2. Isolation: Concurrency Control
    - Serializability and conflict serializability
    - 2PL and Strict 2PL
3. Worksheet

Transaction

- Sequence of instructions you want to execute as if one:

```
CookieJar := CookieJar - 1
CookiesAted := CookiesAted + 1
```

- Want to ensure:
  - Either both execute, or neither.
  - No other transaction sees only part of this execution:

ACID

- Set of formal properties a transaction has:
  - Atomicity – all or nothing / commit or abort
    - Take cookie and eat it, or don’t take cookie
  - Consistency – database remains in consistent state afterwards
    - CookiesInJar >= 0
  - Isolation – transaction runs as if it’s the only one
    - CookiesInJar + CookiesAted == max
  - Durability – changes are never lost
    - CookieMonsterApp crashes, wakes back up, CookiesinJar and CookiesAted are whatever they were before.

Isolation - Concurrency Control

- Could just execute 1 transaction at a time…
  - Slow!
- Want to maximize parallelism while maintaining a sense of isolation.
  - “Concurrency Control!”

Serializability

- Serial schedule – “run one at a time”
- Serializable schedule – schedule whose outcomes are equivalent to a serial schedule.

Time	Transaction 1	Transaction 2
1		CookiesBaking := 5
2		Jar := Jar + CookiesBaking
3	Jar := Jar – 1	
4	Ated := Ated + 1	

Serial, serializable, or neither?

Serializability Disclaimer

- Serializable in databases:
  - From Wikipedia: “in concurrency control of databases, a transaction schedule is serializable if its outcome is equal to the outcome of of its transactions executed serially”
- Serialization in rest of CS:
  - In the context of data storage and transmission, serialization is the process of translating data structures or object state into a format that can be stored.

Serializability

- Serial schedule – “run one at a time”
- Serializable schedule – schedule whose outcomes are equivalent to a serial schedule.

Time	Transaction 1	Transaction 2
1		CookiesBaking := 5
2	Jar := Jar – 1	
3	Ated := Ated + 1	
4		Jar := Jar + CookiesBaking

Serial, serializable, or neither?

Schedule Abstraction

- Talking about the actual semantics of a program becomes hard
  - instead, we just simplify to Reads and Writes.

Time	Transaction 1	Transaction 2
1		CookiesBaking := 5
2	Jar := Jar – 1	
3	Ated := Ated + 1	
4		Jar := Jar + CookiesBaking

Schedule Abstraction

Time	Transaction 1	Transaction 2
1		CookiesBaking := 5
2	Jar := Jar – 1	
3	Ated := Ated + 1	
4		Jar := Jar + CookiesBaking

Time	Transaction 1	Transaction 2
1		W(CookiesBaking)
2	R(Jar)	
3	W(Jar)	
4	R(Ated)	
5	W(Ated)	
6		R(Jar)
7		W(Jar)

Serializability

- Is this schedule serializable?
  - No!
  - How can we tell in general?

Time	Transaction 1	Transaction 2
1		W(CookiesBaking)
2		R(Jar)
3	R(Jar)	
4	W(Jar)	
5	R(Ated)	
6	W(Ated)	
7		W(Jar)

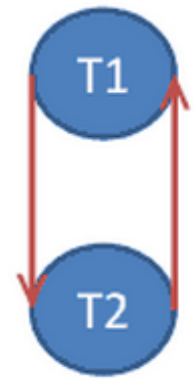
Conflicts

- Conflict:
  – two operations in
  – different transactions on
  – the same object where
  – **at least one is a Write**.
- **Concurrency issues can only arise in the face of conflicts.**

How many conflicts here?

Time	Transaction 1	Transaction 2
1		W(CookiesBaking)
2		R(Jar)
3	R(Jar)	
4	W(Jar)	
5	R(Ated)	
6	W(Ated)	
7		W(Jar)

Time	Transaction 1	Transaction 2
1		W(CookiesBaking)
2		R(Jar)
3	R(Jar)	
4	W(Jar)	
5	R(Ated)	
6	W(Ated)	
7		W(Jar)



Time	Transaction 1	Transaction 2
1		W(CookiesBaking)
2	R(Jar)	
3	W(Jar)	
4	R(Ated)	
5	W(Ated)	
6		R(Jar)
7		W(Jar)

Conflict Serializability

- A schedule is “conflict serializable” if it is equivalent to some **serial schedule with the conflicts in the same order**.
  - A schedule is conflict serializable if and only if its dependency graph is acyclic.
  - Another way to look at it: Sliding operations.

Serializability vs. Conflict Serializability

- Why not just check for serializability?
  - We can verify conflict serializability using dependency graph; much harder to verify serializability.
- Any conflict serializable schedule is serializable.
- Some serializable schedules are not conflict serializable.
- This means we throw out valid schedules!
  - That’s ok – it’s the price we pay.
- Questions?

Locks

- We use locks to control access to objects.
  - Shared lock: multiple transactions can have a shared lock on the same item (e.g., reading)
  - Exclusive lock: only one (and no other lock) on this item (e.g., writing)

Time	Transaction 1	Transaction 2
1	Lock_X(Jar)	Lock_X(CookiesBaking)
2	R(Jar)	W(CookiesBaking)
3	W(Jar)	Unlock(CookiesBaking)
4	Unlock(Jar)	
5	Lock_X(Ated)	Lock_X(Jar)
6	R(Ated)	R(Jar)
7	W(Ated)	W(Jar)
8	Unlock(Ated)	Unlock(Jar)

Two-Phase Locking (2PL)

- We add a rule for how a transaction may acquire locks:
  - Once you release a lock, you may never acquire a new lock.
- Ensures conflict serializability!
  - Wut, really?
  - In order for a conflict cycle to occur, we need to release a lock so other guy can use our object, then acquire a lock to use the other guy’s object

Cascading Aborts and Strict 2PL

- What happens here?

  T1: R(A), W(A), Abort

  T2: R(A), W(A)
- How to fix?
- Strict Two-Phase Locking:
  - All locks held by transaction are only released at the end of the transaction.

Deadlocks

- Dealing with deadlocks:
  - Prevention – stop them from occurring
  - Detection – stop them while occurring
  - In practice: timer

Time	Transaction 1	Transaction 2
1	Lock_X(A) (granted)	
2		Lock_X(B) (granted)
3	Lock_X(B) (waiting)	
4		Lock_X(A) (waiting)
5	…	…

Deadlock Prevention

- Disallow deadlocks from ever occurring.
- Two transactions T_{old} and T_{young}.
  - Wait-Die:
    - If T_{old} is waiting for a lock from T_{young}, he just waits.
    - If T_{young} is waiting for a lock from T_{old}, he kills himself.
  - Wound-Wait:
    - If T_{old} is waiting for a lock from T_{young}, he kills T_{young}.
    - If T_{young} is waiting for a lock from T_{old}, he just waits.
- If you die, you restart with ***original*** timestamp.

Deadlock Detection

- Waits-for graph of all transactions.
- If cycle exists, shoot one of the transactions in the cycle.

Summary

- For isolation, we need a serializable schedule.
- Strict 2PL gives us conflict serializability automatically, and avoids cascading aborts.
- Can either detect deadlocks using waits-for graph or prevent it using wait-for or wound-wait

Worksheet!