Lecture 13:
Continuing Transactions

Wednesday, March 4, 2015

Agenda

- Discuss any issues with HW #3
- Continue transactions
- Work on project proposal (Checkpoint #2)

Review: ACID Properties

- **Atomicity** = A tx's operations either happen in their entirely or not at all. There are only two outcomes (commit or rollback).
- **Consistency** = If the database satisfies the constraints at the beginning of the tx, and if the application is written correctly, then the constraints must hold at the end of the tx. The duty of the tx is to ensure that the database remains consistent.
- **Isolation** = Although a tx can be interleaved with other txs, it executes in isolation. There is no interference from other txs.
- **Durability** = Txs have to persist their updates to disk (not just main memory).

Transaction Definitions

- **From the application-level:**
  -- A transaction = one or more operations, which represents a logical unit of work. This logical unit of work cannot be broken up into smaller units without potentially compromising the integrity of the database.

- **For the DBMS-level:**
  -- A transaction = a sequence of read and/or write operations

Concurrency Control

- **The Problem**:
  -- A transaction can perform many updates
  -- For efficiency reasons, we can't wait for one tx to complete before starting another tx
  -- How can we allow txs to be interleaved without letting them hurt one another?

- **The Solution:**
  -- Use a scheduler to decide which tx goes next
  -- A schedule is a sequence of interleaved actions from all transactions
  -- Our goal is to understand what makes a good schedule

Serializable Schedule

A schedule is *serializable* if it is equivalent to a serial schedule

Running example

T1
READ(A, x)
x := x+100
WRITE(A, x)
READ(B, x)
x := x+100
WRITE(B,x)

T2
READ(A, y)
y := y*2
WRITE(A,y)
READ(B,y)
y := y*2
WRITE(B,y)

Want to construct a schedule for these two independent txs, T1 and T2

A Serial Schedule

T1	T2
READ(A, x)	
x := x+100	
WRITE(A, x)	
READ(B, x)	
x := x+100	
WRITE(B,x)	
	READ(A,y)
	y := y*2
	WRITE(A,y)
	READ(B,y)
	y := y*2
	WRITE(B,y)

A Serializable Schedule

T1	T2
READ(A, x)	
x := x+100	
WRITE(A, x)	
	READ(A,y)
	y := y*2
	WRITE(A,y)
READ(B, x)	
x := x+100	
WRITE(B,x)	
	READ(B,y)
	y := y*2
	WRITE(B,y)

Question: what efficiencies do we gain with this schedule?

Another Serializable Schedule

T1	T2
READ(A, x)	
x := x+100	
WRITE(A, x)	
READ(B, x)	**READ**(A,y)
x := x+100	y := y*2
WRITE(B,x)	**WRITE**(A,y)
	READ(B,y)
	y := y*2
	WRITE(B,y)

A Non-Serializable Schedule

T1	**T2**
READ(A, x)	
x := x+100	
WRITE(A, x)	
	READ(A,y)
	y := y*2
	WRITE(A,y)
	READ(B,y)
	y := y*2
	WRITE(B,y)
READ(B, x)	
x := x+100	
WRITE(B,x)	

Conflicting actions

Two actions belonging to same tx: $r_1(X); w_1(Y)$

Two writes by T_1 and T_2 to same record: $w_1(X); w_2(X)$

Read/write by T_1 and T_2 to same record: $w_1(X); r_2(X)$

$r_1(X); w_2(X)$

In other words, two actions *conflict* if they involve the same record and at least one of them is a write.

A *serializable* schedule is derived by swapping the non-conflicting actions of multiple concurrent txs (e.g. reads on the same record, reads and writes on different records).

[TX DEMO]

Reflections

- The demo showed that when tx T2 tried to update the same record as tx T1, T2's update hung until T1's commit.

  Question: Is this behavior expected?

  Ans: Yes, this shows the effects of serializing writes to the same record.

Reflections

- The demo showed that during an update by T1, T2 was able to read the same record that T1 was modifying.

  Question: Which value of the record did T2 read?

  Ans: T2 read the last committed value of the record (not the dirty value that T1 was actively modifying).

- Observation: T2's read did not hang the way it did during the conflicting write. This implies that write-read conflicts are resolved by the DBMS. How?

  Ans: This is done using Multiversion Concurrency Control (MVCC).

  Question: what side-effects can MVCC have?

Rolled-back Transactions

- **Rollbacks initiated by application**:
  -- Rollback when user cancels an operation
  -- Rollback if one or more constraints are not satisfied

- **Rollbacks initiated by DBMS:**
  -- Rollback when database aborts
  -- Rollback when there is a deadlock condition
  -- Rollback when there is a timeout

- **Schedules with Rolled-back Transactions**
  -- When a tx rolls back, the recovery manager undoes its updates
  -- But some of its updates may have affected other txs!

Issues with Rollback

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
	R(B)
	W(B)
	Commit
Rollback	

Recoverable Schedule

A schedule is *recoverable* if:

- It is *serializable*
- Whenever a tx T commits, all txs that have written records read by T have already committed

Non-Recoverable and Recoverable Schedules

Schedule A: non-recoverable

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
	R(B)
	W(B)
	Commit
Rollback	

Schedule B: recoverable

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
	R(B)
	W(B)
	Rollback
Rollback	

Optional References

- Jim Gray and Andreas Reuter. Transaction Processing: Concepts and Techniques. Morgan Kaufmann, 1993.

- Philip Bernstein et al. Concurrency Control and Recovery in Database Systems. Addison-Wesley, 1987.

Checkpoint 1: Project Groups: Done!

Grp	Members
1	Matthew Egbom, Jewel Langevine, and Lerone Williams
2	Nathan Waters and Nur Ridzuan
3	Steve Franklin, Sadie Sublousky, and Tien-Yu Huang
4	Mills Hill
5	Alexander Crompton and Jacob Rachiele
6	Mitali Sathaye
7	Nikolaj Plagborg-Moller and Fabiana Latorre
8	Hannah Jane DeCiutiis, Kathryn McDermott, and Esther Schenau
9	Khang Pham and Don Pham
10	Alexia Mercado and Cyndia Munoz
11	Thomas Johnson and John Loftin
12	Ross Yudkin, Kurt Probe, and Andrew Chang-Gu
13	Tianxiang Zhang, Xiaolin Lu, and Happy Situ
14	Kaitlin Vanderlaan, Julia Haschke, and Sarah Luna
15	Brian Huang, Sergio Mier, and Jun-Bo Shim
16	Jose Cortez, David Hernandez, and Tara Woolheater
17	Kerri Grier and Chris Oballe
18	Matthew Jones, Thomas Reay, and Brooke Noble
19	Seata Moji and Alexander Thola
20	Hyun Seo and Parth Patel
21	Yifang Peng and Jiannan Zhang
22	Dustin Dies, Sreejon Sen, and Huynh Lam
23	Cameron Miller, Jorge Paramo, and Kyle Kerr
24	Humza Rashid, Mark Slater, and Matthew Mcnair
25	Robert Mcneil and Zachary Williams
26	Bailey Lund, Kristine Chen, and Irene Jea
27	Damilola Shonaike and Bryan Landes

Checkpoint 2: Project Proposal

- **Due today** (Wednesday, 03/04)
- Should be about **1 page** in length.
- Suggested content:
  - -title and group members
  - -short description of the project
  - -list any interesting issues or unanswered questions
  - -expected responsibilities/deliverables for each group member
  - -**important:** tools and datasets you are planning to use
- Submit in class or by email

Next Class

- Continue transactions
- Work on project checkpoints 3 and 4