Weekly Research Progress Report

Student: Sungho Hong **Date:**6/12/2020 **# of hrs worked this week:** 30

Problems discussed in last week's report

1. **Categorizing test cases**
    a. Test cases for RC and SI
    b. Reference materials for test-cases of RC and SI
2. **Reference materials**
    a. Making snapshot isolation serializable, ACM Transactions on DB systems, 2005
    b. A Read-Only Transaction Anomaly Under Snapshot Isolation, SIGMOD, 2004
    c. A Critique of ANSI SQL Isolation Levels, SIGMOD, 1995

Test cases for RC and SI

O : Preventable **X** : Unavoidable

Test ID	P0	P1	P2	P3	P4	P5	P6
Isolation	Dirty Read	Dirty Write	Fuzzy Read	Lost update	Read Skew	Write Skew	Read Anomaly
RC	O	O	X	X	X	X	X
SI	O	O	O	O	O	X	X
RC + SSN	X	X	O	O	O	O	O

P0: Dirty Read
T1 has read a data item that was never committed and so never really existed.
One of the transactions is expected to abort in this scenario.

		5	**10**	**15**	**20**
T0	(X=0)	W(X=10)		C or R	
T1			R(X=10)		C or R

		5	**10**	**15**	**20**
T0	(X=0)	W(X=10)			C or R
T1			R(X=10)	C or R	

First: T0 commits and T1 aborts as RC, Second: T0, T1 commits
The reason is kvstore is empty when T1 reads the data X, and the kvstore is still empty when the T0 is trying to add the data X.

P1: Dirty Write
It is unclear what the correct data value should be.
One of the transactions is expected to abort in this scenario.

	5	**10**	**15**	**20**
T0	W(X=10)		C or R	
T1		W(X=20)		C or R

	5	**10**	**15**	**20**
T0	W(X=10)			C or R
T1		W(X=20)	C or R	

First, Second: Both T0 and T1 commits

P2: Fuzzy / Non-Repeatable Read

T0 receives a modified value or discovers that the data item has been deleted.
The first one is expected to abort and the second one is expected to commit.

	5	10	15	20	25
T0	R(X=10)			R(X=20)	C or R
T1		W(X=20 or delete)	C or R		

	5	10	15	20	25
T0	R(X=10)		R(X=10)	C or R	
T1		W(X=20 or delete)			C or R

First: T0 aborts, Second: both T0 and T1 commits

P3: Lost Update

T1's update will be lost. Both of the scenarios contain anti-read dependencies, so the RC+SSN will abort automatically.

	5	10	15	20	25	30
T0	R(X=100)				W(x=130)	C
T1		R(X=100)	W(X=120)	C		

T0 aborts and T1 commits

P4: Read Skew

T1 reads y, it may see an inconsistent state. Both of the scenarios contain anti-read dependencies, so the RC+SSN will abort automatically. Fuzzy Reads is a degenerate form of Read Skew where x=y.

	5	10	15	20	25	30
T0	R(X)				R(Y)	C or A
T1		W(X)	W(Y)	C		

	5	10	15	20	25	30	35	40
T0	R(X=50)						R(y=90)	C

T1	R(x=50) W(x=10) R(y=50) W(y=90) C

P4: Write Skew

Two different data items were updated, each under the assumption that the other remained stable. Both of the scenarios contain anti-read dependencies, so the RC+SSN will abort automatically.

5 10 15 20 25 30 35 40 45 50

T0	R(X) W(Y) C R(X) R(Y) C
T1	R(Y) W(X) C R(X) R(Y) C

T0	R(X) W(Y) C R(X) R(Y) C
T1	R(Y) W(X) C R(X) R(Y) C

5 10 15 20 25 30 35

T0	R(X=50) R(Y=50) W(y=-40) **C**
T1	R(X=50) R(y=50) **C**

5 10 15 20 25 30 35 40

T0	R(X0,70) R(Y0,80) W(X1, -30) C
T1	R(X0,70) R(Y0,80) W(Y2, -20) C

P5: Read-Only Anomaly

it was assumed that read-only transactions always execute serializably, without ever needing to wait or abort because of concurrent update transactions. Read only transaction T3 prints out X = 0 and Y = 20, while final values are Y = 20 and X = -11. The fact that SI allows commit order different than serial order is what causes the anomaly.

5 10 15 20 25 30 35 40 45 50

T0	R(Y0,0) W(Y1,20) C
T1	R(X0,0) R(Y0,0) W(X2, -11) C
T2	R(X0,0) R(Y1,20) C

Reference Materials of RC and SI(MVCC)

Isolation Levels

- trade throughput for correctness
    - Lower isolation levels increase transaction concurrency but risk showing transactions a fuzzy or incorrect database

Transaction

- a set of actions such as Reads and Writes that transform the database from one consistent state to another

History

- models the interleaved execution of a set of transactions as a linear ordering of their actions

Dependency graph

- defining the temporal data flow among transactions.
- Two histories are equivalent if they have the same committed transactions and the same dependency graph

Serializability

- A history is serializable if it is equivalent to a serial history
    - if the history has the same dependency graph (inter-transaction temporal data flow) as some history that executes transactions one at a time in sequence

Concurrency

- if T1 and T2 transactional lifetimes overlap
    - [start(T1), commit(T1)] ∩ [start(T2), commit(T2)] ≠ ∅.
    - writes by concurrent transactions, are not visible to the transaction
    - When Ti is ready to commit, it obeys the First Committer Wins rule

First Committer Wins

- Ti will successfully commit if and only if no concurrent transaction Tk has already committed writes (updates) of rows or index entries that Ti intends to write.