FUTUREWEI Technologies

Weekly Research Progress Report

Student: Sungho Hong **Date:**6/15/2020 **# 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
Expected	RC	O	O	X	X	X	X	X
	SI	O	O	O	O	O	X	X
Tested	RC + SSN	X	X	X	X	X	X	X

Prerequisite of RC + SSN

1. Every read the cStamp of the key item updates to the latest cStamp
   a. Currently, I have to manually update the key value to the latest timestamp when I perform a read operation.
2. The validator commits the transactions ordered by their CTS.
   a. The current version of RC + SSN does not abort the anti-read dependency

PRE_1	**5**	**10**	**10**	**25**
T0	W(X)	C		
T1			R(X)	C

PRE_2	**5**	**20**	**25**	**30**
T0	W(X)		C	
T1		R(X)		A

PRE_3	**5**	**10**	**15**	**20**	**25**
T0	R(X=10)			R(X=20)	A
T1		W(X=20 or delete)	C		

P0: Dirty Read

T1 has read a data item that was never committed.
According to RC, T0 should block T1 from reading the X.
According to SI, T1 should read the previous committed version of X.
According to RC + SSN, the transactions read different values depending on the order of the transaction.CTS.

P0_0		5	10 → 15	15	20
T0	(X=0)	W(X=10)		A	
T1			R(X=10)		C

1. Validator commits T0 and then T1
2. Both T0 and T1 commits because
    a. T0 writes X to the empty keystore
    b. T1 get the Latest cStamp of the X value
    c. T1.pstamp = 0, T1.sstamp = 20
3. T1 read the X which is (technically) not yet committed

P0_1		5	10	15	20
T0	(X=0)	W(X=10)			C or R
T1			R(X=0)	C or R	

1. Validator commits T1 and then T0
2. T1 reads the original version of X

P0_2	5	10	15	20 → 10	25	30
T0	W(X=0)	C				
T1			W(X=10)			C or R
T2				R(X=0)	C or R	

1. Validator commits T1 and then T0
2. T1 reads the original version of X

P1: Dirty Write

It is unclear what the correct data value should be.

According to both RC and SI, one of the transactions is expected to abort in this scenario.

According to RC + SSN, the transaction either aborts or commits depending on the order of the transaction.CTS.

P1_0	**5**	**10**	**15**	**20**
T0	W(X=10)		C or R	
T1		W(X=20)		C or R

1. Validator commits T0 and then T1
2. T1 is committed because
   a. pStamp = 0 → 15 (committed time of T0)
   b. sStamp = 20 (committed time of itself)
   c. pStamp < sStamp == Safe to commit

P1_1	**5**	**10**	**15**	**20**
T0	W(X=10)			C or R
T1		W(X=20)	C or R	

1. Validator commits T1 and then T0
2. T1 is aborted because
   a. pStamp = 0 → 20 (committed time of T0)
   b. sStamp = 15 (committed time of itself)
   c. pStamp > sStamp == abort

P2: Fuzzy / Non-Repeatable Read

T0 receives a modified value or discovers that the data item has been deleted.
According to RC, the transaction cannot avoid fuzzy read.
According to SI, T1 should read the previous committed version of X.
According to RC + SSN, the transactions read different values depending on the order of the transaction.CTS.

P2_0	**5 → 15**	**10**	**15**	**20 → 15**	**25**
T0	R(X=20)			R(X=20)	C or R
T1		W(X=20 or delete)	C or R		

1. Validator commits T1 and then T0
2. T0 reads the committed X twice.

P2_1	**5**	**10**	**15**	**20**	**25**
T0	R(X=10)		R(X=10)	C or R	
T1		W(X=20 or delete)			C or R

1. Validator commits T0 and then T1
2. T1 reads the uncommitted X twice

P3: Lost Update

T1's update will be lost.
According to RC, the transaction cannot avoid lost updates.
According to SI, Either T1 or T0 should be aborted.
According to RC + SSN, the transaction cannot avoid lost updates.

P3_0	**5 → 20**	**10**	**15**	**20**	**25**	**30**
T0	R(X=100)				W(x=130)	C
T1		R(X=100)	W(X=120)	C		

1. Validator commits T1 and then T2
2. T0 overwrites the committed X

P4: Read Skew

T1 reads y, it may see an inconsistent state. 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. This is an inherent issue of SI, and RC + SSN experience the same problem.

P4_0	5	10 → 25	15	20	25	30	35 → 30	40 → 25	45 → 25	50 → 30
T0	R(X=70)		W(Y=80)		C		R(X=80)	R(Y=80)		
T1		R(Y=70)		W(X=80)		C			R(Y=80)	R(X=80)

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 T2 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
  - $[start(T1), commit(T1)] \cap [start(T2), commit(T2)] \neq \varnothing$.
  - 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.