CSCI235 Database Systems

Multiversion Concurrency Control (MVCC)

Dr Janusz R. Getta

School of Computing and Information Technology -
University of Wollongong

Multiversion Concurrency Control

Outline

Principles

Each time a data item is inserted or deleted or updated a new version of such data item is saved in a database

Older versions of data items are kept available for transactions

Each version of data item is stamped with the commit time of the trasaction that created the version

A tuple `(x,v)` in a relationaln table `R(X,V), PK=(X)` is replaced with a versioned tuple `(x, T, v)` where `T` is an identifier of a transaction that created the new tuple ; `(x,T)` is a unique key of versioned tuple

A table `commit-time(transaction-id, commit-timestamp)` keeps information all committed transactions

When a transaction `T'` updates a tuple with a key `x` for the first time in the transaction it is given the most recent version of a tuple `(x, T, v)`

Such tuple must be committed before `T'` can update it

Principles

`T'` creates a new version `(x,T',v')` of the tuple `(x,T,v)`

Any further updates by `T'` on the same tuple `(x,T',v)` occur in-place

If `T'` commits then it leaves the newest version of the tuple with the last update on it

If `T'` does not commit then no new version `(x,T',v')` is left

If `T'` deletes a tuple `(x,T,v)` then a new version `(x,T',⊥)` is created

If `T'` inserts a tuple `(x,T',v')` then such insertion is possible only when a tuple `(x,T',⊥)` exists or when no tuple with a key `x` exists

Multiversion Concurrency Control (MVCC) allows for linear version histories `(x,T1,v1)`, `(x,T2,v2)`, ... , `(x,Tn,vn)` when the transactions committed in the order `T1`, `T2`, ... ,`Tn` and for all `i=2, ... n` a transaction `Ti` has seen a version `(x,Ti-1,vi-1)` when creating a version `(x, Ti, vi)`

Principles

`commit-time` table contains information about commit time of each transaction

MVCC assumes two kinds of transactions: read-only transactions and update transactions

The following execution is not conflict serializable by a single version scheduler

Not conflict serializable execution

T1	T2
	write(x,2)
a = read(x)	
b = read(y)	
	write(y,3)
b = read(y)	

Principles

If `start-time(T1) < start-time(T2)` then MVCC scheduler creates a conflict serializable execution

Conflict serializable execution by MVCC scheduler

```
T1 (read only)          T2
                        write(x|x',2)
a = read(x)
b = read(y)
                         write(y|y',3)
b = read(y)
```

No conflicts because the transactions operate on disjoint sets of data

Multiversion Concurrency Control

Outline

Transaction-level read consistency

Under transaction-level read consistency the read actions of every read-only transaction `T` read the same database version while every update transaction runs at serializable isolation level

Every update transaction operates on the most recent tuple version

To ensure transaction-level read consistency all actions by update transactions are protected by commit-duration locks

Typical anomalies: dirty reads, not repeatable reads and dirty writes

Dirty write means that if a transaction writes, inserts or deletes a tuple with a key `x` then the same tuple with a key `x` is written. inserted or deleted by another active transaction

Read-only transactions cannot do dirty read or not repeatable read because the same read still uses the same old version of data item

Multiversion Concurrency Control

Outline

- Principles
- Transaction-level read consistency
- Statement-level read consistency
- Snapshot isolation (SI) protocol
- Serializable Snapshot Isolation (SSI) protocol

Statement-level read consistency

Under statement-level read consistency the read actions of every read-only transaction read from the start-time version and the update actions of every update transaction operate on the most recent versions of tuples

Read actions of update transactions can be not repeatable and read from the most recent committed version as of the time when the SQL statement that gave rise to the read action was started

It means that all the tuples read by a single SQL statement are read from the same committed version

Transactions that run at statement-level read consistency do not do dirty reads and or dirty writes

Because statement-level read consistency prevents dirty writes and dirty reads it implies SQL isolation level read committed

Multiversion Concurrency Control

Outline

- Principles
- Transaction-level read consistency
- Statement-level read consistency
- Snapshot isolation (SI) protocol
- Serializable Snapshot Isolation (SSI) protocol

Snapshot isolation (SI) protocol

Several well-known database management systems (e.g. Oracle) that use versioning for concurrency control enforce an isolation level called snapshot isolation protocol

Under snapshot isolation protocol no distinction is made between read-only transcations and update transactions

Under snapshot isolation protocol all reads of any transaction are performed on start-time version of the transaction except when read action on data updated by the transaction itself reads the current data

All update actions operate on the most recent versions of data items (tuples)

Under snapshot isolation protocol if two committed transactions `Ti` and `Tj` are concurrent, that is, at some timepoint both are active, then they are required to have the disjoint-write property

It means that the sets of the data items updated by the transactions, must be disjoint, `write-set(Ti) ∩ write-set(Tj) = ∅`

Snapshot isolation (SI) protocol

Under the snapshot-isolation protocol, all reads within a transaction see a consistent view of a database (transaction-level read consistency)

A transaction operates on a private snapshot of the database taken just before its first read

The concurrently running transactions are prohibited from modifying the same data items

It implies no dirty reads, no not repeatable reads and no phantoms

Snapshot isolation protocol allows for other anomalies like write skews where the transactions read the same data and modify disjoint sets of data

Snapshot isolation (SI) protocol

Sample write skew anomaly

Let `x` and `y` be the family bank accounts such that `x + y ≥ 0` and `x = 50$` and `y = 50$` and `80$` is withdrawn from both accounts

Incorrrect processing controlled by SI protocol

T1	T2	x	x'	y	y'	a	b	c	d
a = read(x)		50		50		50			
b = read(y)		50		50		50	50		
	c = read(x)	50		50		50	50	50	
	d = read(y)	50		50		50	50	50	50
if a+b>=80 then write(x,a-80)		50	-30	50		50	50	50	50
	if c+d>=80 then write(y,c-80)	50	-30	50	-30	50	50	50	50

Snapshot isolation (SI) protocol

Sample write skew anomaly

`Computer Science` department offers 2 courses: `CSCI235` and `CSCI251`, one of the courses must be moved to `Mathematics`

Incorrrect processing controlled by SI protocol

```
T1                                         T2

SELECT count(*) INTO x
FROM COURSE
WHERE offered_by = 'Computer Science'

                                           SELECT count(*) INTO y
                                           FROM COURSE
                                           WHERE offered_by = 'Computer Science'

if x >= 2 then UPDATE COURSE
               SET offered_by = 'Mathematics'
               WHERE code = 'CSCI235';

                                           if y >= 2 then UPDATE COURSE
                                                          SET offered_by = 'Mathematics'
                                                          WHERE code = 'CSCI251'
                                                          COMMIT;

COMMIT;
```

Multiversion Concurrency Control

Outline

Serializable Snapshot Isolation (SSI) protocol

Serializable Snapshot Isolation (SSI) protocol runs transactions using SI protocol and performs additional checks to determine whether anomalies are possible

Under SSI protocol transactions that violate serializability are simply aborted

PostgreSQL's SSI implementation uses MVC data as well as a new lock manager to detect conflicts

SI allows for not serializable executions that do not exhibit any of anomalies identified in SQL standard (dirty read, not repeatable read and phantom anomalies)

SSI identifies so called rw-antidependencies: If `Ti` write a new version of a data item and `Tj` reads the previous version of the same data item then `Ti` appears to have executed after `Tj` because `Tj` did not see a new version (rw-conflict)

Serializable Snapshot Isolation (SSI) protocol

Every cycle in the serialization graph (anomaly) contains at least two adjacent rw-antidependencies

SSI detects two adjacent rw-antidependencies in a serialization graph and aborts one of the transactions

References

T. Connoly, C. Begg, Database Systems, A Practical Approach to Design, Implementation, and Management, Chapter 22.5 Concurrency Control and Recovery in Oracle, Pearson Education Ltd, 2015

S. Sippu, E. Soisalon-Soininen, Transaction Processing, Chapter 12 Concurrency Control by Versioning, Springer, 2014

D. R. K. Ports, K. Grittner, Serializable Snapshot Isolation in PostgreSQL, Proceedings of the VLDB Endowment, Vol. 5, No. 12, 2012