6 Multiversion Concurrency Control

6.1 Motivation: Versions and serializability
6.2 MVCC using timestamps
6.3 Two Version MV2PL
6.4 MVCC for read only TA
6.5 Read consistency MVCC – no read locks
6.6 Snapshot isolation

Motivation
r1[x] w1[x] r2[x] w2[y] r1[y] w1[z] c1 w2[a] c2
not serializable.
• If r1[y] had arrived at the scheduler before
  w2[y] the schedule would have been serializable.
Timestamp ordering: scheduler forces serialization
according to TA age, Multiversion protocols?

Multiversion CC: Read a consistent version which
has not been written by an uncommitted TA:
writers of x: prepare a new version x'
readers: read a committed version

Multiversion schedules
r1[x] w1[x] r2[x] w2[y'] r1[y] w1[z] c1 w2[a] c2
T2 produces version y', but T1 read an old,
committed version y.
No conflict any more: T1 read a version of x which
was committed before BOT(T1)

Notation:
A data object x has versions x_i, x_j such that
transactions T_i , T_j … have written (“produced”) the
 corresponding versions

Serializability
Conflict serializability inappropriate:
MC: H_1 = w_0(x_0) c_0 w_1(x_1) w_1(z_1) c_1 r_2(x_0) w_2(y_2) c_2
One copy: H'_1 = w_0(x) c_0 w_1(x) w_1(z) c_1 r_2(x) w_2(y) c_2
Same conflicts on x, but H'_1 does not have the same
conflict properties as H_1:
H_1: T_2 reads from T_0
H'_1: T_0 < T_1 < T_2 since T_2 reads from T_1
Conflict properties different: w_1(x_1), r_2(x_0) not in

Correctness reasoning
Conflict equivalence not appropriate, but…

View equivalence:
Reads-From relation of schedule H is equivalent to
Reads-From relation of some serial schedule S

First attempt:
Scheduler has to produce MV histories H which are view
serializable to some one-version history
(1V history),
e.g. serialization Graph SG(H) is acyclic

Correctness reasoning
Not sufficient to show that MV serialization graph is cycle free!

Example:
w_0(x_0) w_0(y_0) c_0 r_1(x_0) r_1(y_0) w_1(x_1) w_1(y_1) c_1 r_2(x_0) r_2(y_1) c_2
not equivalent to any 1V history, but SG acyclic

T0 T1 T2
1 V history (e.g.)
w_0(x_0) w_0(y_0) c_0 r_2(x_0) r_2(y_1) c_2 r_1(x_1) r_1(y_1) w_1(x_1) w_1(y_1) c_1
T2 reads from T_0, in the MV history from T_0 and from T_1
Correctness reasoning

Problem in the example above:
H: \(w_0(x_0) w_0(y_0)c_0 \ldots w_1(x_1) w_1(y_1) c_1 r_2(x_0) r_2(y_1) c_2\)

T2 reads from T0 but the value read has already been modified by a subsequent TA

An MV history is a 1-serial MV history, if for every read-from triple \((T_j, x, T_i)\), i.e. Ti reads x from Tj, Tj is the last TA preceding Ti which produced any version of x

H not 1-serializable: \((T_0, x, T_2)\) but T1 produced a newer version

MVCC correctness

Deciding if a MV history is 1-serial MV is NP complete.

But there is a polynomial time characterization of "multiversion conflicts"

Intuitively: ri(x) and wk(x) are not in conflict, since k writes a different version than l reads.

Gives rise to an MV conflict graph

MV-conflict serializability \(\Rightarrow\) no cycles

6.2 MV timestamp ordering

Wanted: schedulers which guarantee conflict serializability using multiple versions

MV Timestamp ordering

- each transaction \(T_i\) is assigned a unique timestamp \(ts(t_i)\)
- \(r_i(x)\) is mapped to \(r_i(x_k)\) where \(x_k\) is the version that carries the largest timestamp \(\leq ts(t_i)\)
- \(w_i(x)\) is
  - rejected if there is \(r_i(x_k)\) with \(ts(t_k) < ts(t_i) < ts(t_j)\)
  - mapped into \(w_i(x_k)\) otherwise
- \(c_i\) is delayed until \(c_j\) of all transactions \(T_j\) that have written versions read by \(T_i\)

6.3 MV2PL scheduling

Two Version 2 PL approach

Requirements:
- Exactly one committed version \(x\) exists, at most one version \(x'\) which is being produced (written) \(\Rightarrow\) use write locks for w-w conflicts
- reading of the committed version \(x\) is possible if a new version is produced \(\Rightarrow\) no r-w conflict

Critical situation: commit newly produced version \(x'\) (\(\Rightarrow\) forget old committed version \(x\) – only one may exist!)
MV2PL: conversion

- Conversion of x to the new version x’ has to be done with care:
  - no reader should read x (or x’?) during conversion
  - if reader of x still active, do not convert

Needed: (1) read and write locks
(2) certify lock for the conversion phase

Example 2PL MVCC

Input schedule:
\[ s = r_1(x) w_1(x) r_2(x) r_1(y) w_2(x) c_1 \]

MV2PL output schedule?
\[ r_1(x_0) w_1(x_1) r_2(x_0) r_1(y_0) w_1(y_1) c_1 w_2(x_2) c_2 \]

Example 2PL MVCC

Input schedule:
\[ s = r_1(x) w_1(x) r_2(x) r_1(y) w_2(x) c_1 \]

MV2PL output schedule?
\[ r_1(x_0) w_1(x_1) r_2(x_0) r_1(y_0) w_1(y_1) c_1 w_2(x_2) c_2 \]

Example 2PL MVCC

Input schedule:
\[ s = r_1(x) w_1(x) r_2(x) r_1(y) w_2(x) c_1 \]

MV2PL output schedule?
\[ r_1(x_0) w_1(x_1) r_2(x_0) r_1(y_0) w_1(y_1) c_1 w_2(x_2) c_2 \]

Example 2PL MVCC

Input schedule:
\[ s = r_1(x) w_1(x) r_2(x) r_1(y) w_2(x) c_1 \]

MV2PL output schedule?
\[ r_1(x_0) w_1(x_1) r_2(x_0) r_1(y_0) w_1(y_1) c_1 w_2(x_2) c_2 \]

Example 2PL MVCC

Input schedule:
\[ s = r_1(x) w_1(x) r_2(x) r_1(y) w_2(x) c_1 \]

MV2PL output schedule?
\[ r_1(x_0) w_1(x_1) r_2(x_0) r_1(y_0) w_1(y_1) c_1 w_2(x_2) c_2 \]

Example 2PL MVCC

Input schedule:
\[ s = r_1(x) w_1(x) r_2(x) r_1(y) w_2(x) c_1 \]

MV2PL output schedule?
\[ r_1(x_0) w_1(x_1) r_2(x_0) r_1(y_0) w_1(y_1) c_1 w_2(x_2) c_2 \]
Read only MVCC: solution

- Update transactions make a new version of updated data \( x, y, \ldots \) with a timestamp \( t \) at commit. Version of \( x, y, \ldots \) is \( t \).

- Read-only TA with timestamp \( t \) reads only those values \( v \) with version \( t', t = max \{ t', t' < t \} \) => read the last committed values.

Update TA use conventional 2PL protocol with \( r \) and \( w \) locks.

MVCC / Read Only TAs: Example

call sequence: TA1, TA4 and TA5 are RO
\[ r(x) \quad (x) \quad r(y) \quad r(x) \quad (x) \quad r(y) \quad c(x) \quad r(y) \quad c(x) \quad c(y) \quad c(x) \quad r(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \quad c(y) \quad c(x) \Quad \ 2PL \ protocol \ with \ r \ and \ w \ locks.

MVCC: How to implement versions?

Read Only Multiple version CC (used in Oracle)

**No read locks needed for consistent read**

Data have to be temporarily stored anyway: System has to be prepared for Rollback

Implementing versions

PostgreSQL

- do not delete!
- update = insert (append) of a new version!
- erase old versions which are not needed any more.
  (which ones?) – called vacuum clean in PG_SQL

6.5 Read Consistency MVCC

- Combine Read-only TA and lock based cc
  - Read-only TA as above
  - write (x):
    write lock the most current version of x and produce version (x, cts),
    other writers have to wait
  - read (x):
    read last committed version without locking()
    READ COMMITTED, not repeatable

Read Consistency MVCC (2)

- Most significant! No Read locks at all!
- More than READ COMMITTED ... since READ ONLY TA serializable
  - Fits to standard 2PL for R/O transaction
  - but...
    no repeatable read, not serializable

Can we do better?
Even no write locks??
6.6 SNAPSHOT Isolation

'writes' are the problem – of course.

Suppose: w0(x0), c0, r1(x0) r2(x0) w1(x1) c1 w2(x2) c2

How can lost update be prevented?

Avoid conflicting writes of concurrent transactions!

⇒ Write set of concurrent (overlapping!) transactions must be disjoint.

SNAPSHOT isolation

"First commit wins" implementation.

Transaction T:

1. make updates locally (like optimistic cc)
2. Commit step 1:
   validate: have all updated objects the same
   version number which T read? condition (*)
3. If yes: commit else abort

   No writes locks, no read locks!!

SNAPSHOT violation: locking

else: TA1 locks x 2PL if it wants to produce a new
version.

if x already (write) locked by TA* TA1 waits until:
   TA* commits ⇒ TA1 aborts
else
   TA* aborts ⇒ TA1 commits
else commit.

• No read locks needed
• Compatible with update in place, if version reconstructed
  from the log.

SNAPSHOT isolation

• READs: version that was current (last committed) when
  TA started
  e.g. max (xj, ctsj ), ctsj < ts(TA) (*)
  ⇒ transaction level consistent, no read locks

• if write set of TAi und TAj not disjoint:
  abort one of them!

How to implement with / without(!) write locks??

SNAPSHOT isolation

Lock based implementation

Let snapshot number (*) of TA1 be s
TA1: write (x)

if s < current version of x: abort
Some TA* modified x after BOT(TA1) and
committed!

expl: r1(y0) r2(x0) w2(x2) c2 r1(x0) w1(x1)
TA1 aborts
else...

(*)Version that was current when TA started (or: read for the first time)

Summary

MV concurrency control protocols are highly relevant from a practical point of view

• Standard 2V MV2PL and MV TO favor reads
• Read locks in 2V MV2PL needed to protect the
  consistent version read by a TA (certify lock not granted)
• Read only transactions without locks together with 2PL is
  standard in serious DBS
• Snapshot isolation: may be implemented without any
  locks.