Concurrency control
(OCC and MVCC)

COS 418/518
Lecture 16
Michael Freedman & Wyatt Lloyd

Serializability
Execution of a set of transactions over multiple items is equivalent to some serial execution of txns

Lock-based concurrency control

- **Big Global Lock**: Results in a serial transaction schedule at the cost of performance
- **Two-phase locking with finer-grain locks:**
  - Growing phase when txn acquires locks
  - Shrinking phase when txn releases locks (typically commit)
  - Allows txn to execute concurrently, improving performance

Q: What if access patterns rarely, if ever, conflict?
Be optimistic!

- Goal: Low overhead for non-conflicting txns
- Assume success!
  - Process transaction as if would succeed
  - Check for serializability only at commit time
  - If fails, abort transaction
- Optimistic Concurrency Control (OCC)
  - Higher performance when few conflicts vs. locking
  - Lower performance when many conflicts vs. locking

2PL vs OCC

- From "Rococo" paper in OSDI 2014. Focus on 2PL vs. OCC.
- Observe OCC better when write rate lower (fewer conflicts), worse than 2PL with write rate higher (more conflicts)

OCC: Three-phase approach

- Begin: Record timestamp marking the transaction’s beginning
- Modify phase:
  - Txn can read values of committed data items
  - Updates only to local copies (versions) of items (in db cache)
- Validate phase
- Commit phase
  - If validates, transaction’s updates applied to DB
  - Otherwise, transaction restarted
  - Care must be taken to avoid “TOCTTOU” issues

OCC: Why validation is necessary

- New txn creates shadow copies of P and Q
- P and Q’s copies at inconsistent state

When commits txn updates, create new versions at some timestamp t
OCC: Validate Phase

- Transaction is about to commit.
  System must ensure:
  - Initial consistency: Versions of accessed objects at start consistent
  - No conflicting concurrency: No other txn has committed an operation at object that conflicts with one of this txn’s invocations

OCC: Validate Phase

- Validation needed by transaction T to commit:
  - For all other txns O either committed or in validation phase, one of following holds:
    A. O completes commit before T starts modify
    B. T starts commit after O completes commit, and ReadSet T and WriteSet O are disjoint
    C. Both ReadSet T and WriteSet T are disjoint from WriteSet O, and O completes modify phase.
  - When validating T, first check (A), then (B), then (C).
  If all fail, validation fails and T aborted

2PL & OCC = strict serialization

- Provides semantics as if only one transaction was running on DB at time, in serial order
  + Real-time guarantees
- 2PL: Pessimistically get all the locks first
- OCC: Optimistically create copies, but then recheck all read + written items before commit

Multi-version concurrency control

Generalize use of multiple versions of objects
Multi-version concurrency control

- Maintain multiple versions of objects, each with own timestamp. Allocate correct version to reads.
- Prior example of MVCC:

MVCC Intuition

- Split transaction into read set and write set
  - All reads execute as if one "snapshot"
  - All writes execute as if one later "snapshot"
- Yields snapshot isolation < serializability

Timestamps in MVCC

- Transactions are assigned timestamps, which may get assigned to objects those txns read/write
- Every object version \( O_v \) has both read and write TS
  - ReadTS: Largest timestamp oftxn that reads \( O_v \)
  - WriteTS: Timestamp oftxn that wrote \( O_v \)
Executing transaction T in MVCC

- Find version of object O to read:
  - # Determine the last version written before read snapshot time
  - Find O_v s.t. max { WriteTS(O_v) | WriteTS(O_v) <= TS(T) }
  - ReadTS(O_v) = max(TS(T), ReadTS(O_v))
  - Return O_v to T
- Perform write of object O or abort if conflicting:
  - Find O_v s.t. max { WriteTS(O_v) | WriteTS(O_v) <= TS(T) }
  - # Abort if another T' exists and has read O after T
  - If ReadTS(O_v) > TS(T)
    - Abort and roll-back T
  - Else
    - Create new version O_W
    - Set ReadTS(O_W) = WriteTS(O_W) = TS(T)

Digging deeper

1. Notation

W(1) = 3: Write creates version 1 with WriteTS = 3
R(1) = 3: Read of version 1 returns timestamp 3

2. Notation

W(1) = 3: Write creates version 1 with WriteTS = 3
R(1) = 3: Read of version 1 returns timestamp 3

3. Notation

W(1) = 3: Write creates version 1 with WriteTS = 3
R(1) = 3: Read of version 1 returns timestamp 3

4. Notation

W(1) = 3: Write creates version 1 with WriteTS = 3
R(1) = 3: Read of version 1 returns timestamp 3

Find v such that max WriteTS(v) <= (TS = 4)
⇒ v = 1 has (WriteTS = 3) <= 4
If ReadTS(1) > 4, abort
⇒ 3 > 4: false
Otherwise, write object
Digging deeper

Notation

W(1) = 3: Write creates version 1 with WriteTS = 3
R(1) = 3: Read of version 1 returns timestamp 3

W(1) = 3  W(2) = 5
R(1) = 3  R(2) = 5

Find v such that max WriteTS(v) <= (TS = 4)
⇒ v = 1 has (WriteTS = 3) <= 4
If ReadTS(1) > 4, abort
⇒ 3 > 4: false
Otherwise, write object

BEGIN Transaction
tmp = READ(O)
WRITE (O, tmp + 1)
END Transaction

Digging deeper

Notation

W(1) = 3: Write creates version 1 with WriteTS = 3
R(1) = 3: Read of version 1 returns timestamp 3

W(2) = 5
R(2) = 5

Find v such that max WriteTS(v) <= (TS = 5)
⇒ v = 1 has (WriteTS = 3) <= 5
Set R(1) = max(5, R(1)) = 5

write(O) by TS = 4

If ReadTS(1) > 4, abort
⇒ 5 > 4: true
Digging deeper

Notation

\begin{align*}
W(1) &= 3: \text{Write creates version 1 with WriteTS} = 3 \\
R(1) &= 3: \text{Read of version 1 returns timestamp 3}
\end{align*}

\begin{align*}
W(2) &= 5 \\
R(2) &= 5
\end{align*}

BEGIN Transaction
\begin{align*}
tmp &= \text{READ}(O) \\
\text{WRITE}(P, tmp + 1)
\end{align*}
END Transaction

Find \(v\) such that \(\max\{\text{WriteTS}(v)\} \leq (TS = 4)\)
\(\Rightarrow v = 1\) has \((\text{WriteTS} = 3) \leq 4\)
Set \(R(1) = \max(4, R(1)) = 5\)
Then write on \(P\) succeeds as well