CS 448 Database Systems

Two Phase Locking
How do we ensure Serializability

• This is the task of the scheduler.
• There are two basic techniques:
  – Locking
  – Time-Stamp Ordering
• Locking enforces serializability by ensuring that no two txns access conflicting objects in an “incorrect” order.
• Time-Stamp ordering assigns a fixed order for every pair of txns and ensures that conflicting accesses are made in that order.
Two Phase Locking

- Basic 2PL
- Each object has associated with it a lock.
- An appropriate lock must be acquired before a txn accesses the object.
- There are 2 basic types of locks: shared (read) and exclusive (write).
- Two locks, $pl_i[x]$ and $ql_j[y]$, conflict if $x=y$ and $i<>j$; and $p$ and $q$ are conflicting operations.
- 2PL is defined by 3 rules
2 Phase Locking

1. To grant a lock, the scheduler checks if a conflicting lock has already been assigned, if so, delay, otherwise set lock and grant it.

2. A lock cannot be released at least until the DM acknowledges that the operation has been performed.

3. Once the scheduler releases a lock for a txn, it may not subsequently acquire any more locks (on any item) for that txn.
Example

- $T_1 = r_1[x] w_1[y] c_1$
- $T_2 = w_2[x] w_2[y] c_2$
- $rl_1[x] r_1[x] ru_1[x] wl_2[x] w_2[x] wl_2[y] w_2[y] wu_2[x] wu_2[y] c_2 wl_1[y] w_1[y] wu_1[y] c_1$
- This is not SR ($r_1[x] < w_2[x]$ and $w_2[y] < w_1[y]$).
- This is prevented by rule 3.
Deadlocks

• 2PL suffers from the problem of deadlocks.
• \( rl_1[x] \ r_1[x] \ wl_2[y] \ w_2[y] \) followed by TM receiving \( w_2[x] \) and \( w_1[y] \).
• Also due to *lock conversion*: changing a read lock to a write lock – can’t release the lock.
  – Why?
  – What if two txns try to convert at the same time?
2PL ensures Serializability

• Add the lock and unlock operations to the notion of histories.

• Proposition 1: Let $H$ be a history produced by a 2PL scheduler. If $o_i[x]$ is in $C(H)$, then $ol_i[x]$ and $ou_i[x]$ are in $C(H)$, and $ol_i[x] < o_i[x] < ou_i[x]$.

• Proposition 2: Let $H$ be a history produced by a 2PL scheduler. If $p_i[x]$ and $q_j[x]$ ($i<>j$) are conflicting operation in $C(H)$, then either $pu_i[x] < ql_j[x]$ or $qu_j[x] < pl_i[x]$. 


Correctness of 2PL

• Proposition 3: Let $H$ be a complete history produced by a 2PL scheduler. If $p_i[x]$ and $q_i[y]$ are in $C(H)$, then $pl_i[x] < qu_i[y]$.

• Lemma 4: Let $H$ be a 2PL history, and suppose $T_i \rightarrow T_j$ is in $SG(H)$. Then, for some data item $x$, and some conflicting operations $p_i[x]$ and $q_j[x]$ in $H$, $pu_i[x] < ql_j[x]$

• Proof: trivial.
Correctness of 2PL

• **Lemma 5:** Let $H$ be a 2PL history, and let $T_1 \rightarrow T_2 \rightarrow \ldots \rightarrow T_n$ be a path in $SG(H)$, where $n > 1$. Then, for some data items $x$ and $y$, and some operations $p_1[x]$ and $q_n[y]$ in $H$, $pu_1[x] < ql_n[y]$.

• **Proof:** by induction on $n$.

• **Base Case,** $n=2$. Follows from Lemma 4.

• **Induction Step.** Assume true for $n=k$ for $k \geq 2$. By the induction hypothesis, there exist data items $x$ and $z$, and operations $p_l[x]$ and $o_k[z]$ in $H$, such that $pu_l[x] < ol_k[z]$.

• By $T_k \rightarrow T_{k+1}$ and Lemma 4, there exists $y$ and conflicting operations $o'_k[y]$ and $q_{k+1}[y]$ in $H$, such that $o'u_k[y] < ql_{k+1}[y]$. 


Correctness of 2PL

• By proposition 3, $ol_k[z] < o'u_k[y]$. Thus by transitivity, $pu_1[x] < ql_{k+1}[y]$.

• **Theorem:** Every 2PL history $H$ is serializable.

• **Proof:** Suppose, by contradiction, that $SG(H)$ contains a cycle $T_1 \rightarrow T_2 \rightarrow \ldots \rightarrow T_n$, where $n > 1$.

• By Lemma 5, for some data items $x$ and $y$, and some operations $p_1[x]$ and $q_1[y]$ in $H$, $pu_1[x] < ql_1[y]$.

• This contradicts Prop 3. Thus $SG(H)$ is acyclic.
Deadlocks

- 2PL suffers from deadlocks
- Timeouts
- **Waits-for-graph**
  - nodes are transactions
  - add edge $T_i \rightarrow T_j$ whenever $T_i$ waits for a lock held by $T_j$
  - remove an edge when last blocking lock is released
  - a cycle implies a deadlock
  - all cycles need to be broken by choosing a victim txn
Types of Schedulers

• Schedulers can delay, reject, or immediately schedule the operations.
• **Aggressive schedulers** try to avoid delaying operations -- may have to abort later
• **Conservative schedulers** try to avoid aborting by delaying and reordering operations
• Trade-off: depends upon degree of conflict between transactions.
• Conservative schedulers try to anticipate future access of transactions.
Conservative 2PL

- 2PL aborts txns only because of deadlocks.
- Conservative 2PL eliminates deadlocks.
- Each txn predeclares all its operations.
- The scheduler sets all locks of a txn in one step, if it cannot (because there is some conflicting lock), the txn is put in a queue.
- When a lock is released the scheduler checks to see which txns can now acquire all their locks.
- Predeclaring may be difficult or even impossible.
Strict 2PL

• A transaction’s locks are all released together after the DM acknowledges the processing of the transaction’s commit or abort.

• Why?
  – To ensure a strict execution
  – Earliest time at which the scheduler is certain that no more locks will be required by the transaction. Why?
Timing of Lock Release

• Let $H$ be a history produced by a strict 2PL scheduler.
• Suppose $w_i[x] < o_j[x]$.
• By rule 1 of 2PL we must have
  1. $wl_i[x] < w_i[x] < wu_i[x]$, and
  2. $ol_j[x] < o_j[x] < ou_j[x]$
• Because $wl_i[x]$ and $ol_j[x]$ conflict we must have either $wu_i[x] < ol_j[x]$ or $ou_j[x] < wl_i[x]$ (Prop. 2)
Timing of Lock Release

- \( ou_j[x] < wl_i[x] \) with above two is impossible, so we must have: 3. \( wu_i[x] < ol_j[x] \).
- Since \( H \) is produced by a strict 2PL scheduler, we must have: 4. Either \( a_i < wu_i[x] \) or \( c_i < wu_i[x] \).
- From 2, 3, & 4: either \( a_i < o_j[x] \) or \( c_i < o_j[x] \), proving that \( H \) is strict.
- Note that read locks can be released upon termination.