Lecture 7: View-Serializable Schedules
Professor Chen Li

**View Serializability**

<table>
<thead>
<tr>
<th>Conflict equivalent</th>
<th>View equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conflict serializable</td>
<td>View serializable</td>
</tr>
</tbody>
</table>

**Motivating example**

Schedule Q

T1 | T2 | T3
---|----|----
Read(A) | Write(A) | Write(A)
Write(A) | Write(A) |

Same as

\[ Q = r_1(A) w_2(A) w_3(A) \]

P(Q): T1 ⊳ T2

\[ T_3 \]

- Not conflict serializable!
But now compare Q to Ss, a serial schedule:

<table>
<thead>
<tr>
<th></th>
<th>T₁</th>
<th>T₂</th>
<th>T₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>Read(A)</td>
<td>Write(A)</td>
<td>Write(A)</td>
</tr>
<tr>
<td>Ss</td>
<td>Read(A)</td>
<td>Write(A)</td>
<td>Write(A)</td>
</tr>
</tbody>
</table>

- T₁ reads same thing in Q, Ss
- T₂, T₃ read same thing (nothing?)
- After Q or Ss, DB is left in same state

So what is wrong with Q?

---

**Definition** Schedules S₁, S₂ are View Equivalent if:

1. If in S₁: \( w_j(A) \Rightarrow r_i(A) \) then in S₂: \( w_j(A) \Rightarrow r_i(A) \)
2. If in S₁: \( r_i(A) \) reads initial DB value, then in S₂: \( r_i(A) \) also reads initial DB value
3. If in S₁: \( T_i \) does last write on A, then in S₂: \( T_i \) also does last write on A

**Definition** Schedule S₁ is View Serializable if it is view equivalent to some serial schedule
View Serializable $\Rightarrow$ Conflict Serializable
- View Serializable $\neq$ Conflict Serializable
e.g., See Schedule Q
- Conflict Serializable $\Rightarrow$ View Serializable

Lemma
Conflict Serializable $\Rightarrow$ View Serializable

Proof:
Swapping non-conflicting actions does not change what transactions read nor final DB state

Note: All view serializable schedules that are not conflict serializable, involve useless write

$$S = W_2(A) \ldots W_3(A) \ldots$$

no reads
How do we test for view-serializability?


\[ \Rightarrow \] P(S) not good enough...
(see schedule Q)

- One problem: some swaps involving conflicting actions are OK... e.g.:

\[ S = \ldots w_2(A) \ldots r_1(A) \ldots w_3(A) \ldots w_4(A) \]

this action can move
if \( w_4(A) \) exists

- Another problem: useless writes

\[ S = \ldots W_2(A) \ldots W_1(A) \ldots \]

no A reads

To check if \( S \) is View Serializable

Polygraph:
(1) Add final transaction \( T_f \) that reads all DB
(e-eliminates condition 3 of V-S definition)

E.g.: \( S = \ldots W_1(A) \ldots W_2(A) \ldots r(A) \ldots \)

last A write
add
(2) Add initial transaction \( T_b \) that writes all DB (eliminates condition 2 of V-S definition)

\[
E.g.: \quad S = w_1(A) \ldots w_2(A) \ldots 
\]

(3) Create labeled precedence graph of \( S \):

(3a) If \( w_i(A) \Rightarrow \eta(A) \) in \( S \), add \( T_i \rightarrow T_j \)

(3b) For each \( w_i(A) \Rightarrow \eta(A) \) do

\[
\text{consider each } w_k(A): [T_k \neq T_b]
\]

- If \( T_i \neq T_b \land T_j \neq T_f \) then insert \( T_k \rightarrow T_i \) some new \( p \)
- If \( T_i = T_b \land T_j \neq T_f \) then insert \( T_j \rightarrow T_k \)
- If \( T_i \neq T_b \land T_j = T_f \) then insert \( T_k \rightarrow T_i \)

(4) Check if \( LP(S) \) is "acyclic" (if so, \( S \) is V-S)

- For each pair of "\( p \)" arcs \( (p \neq 0) \), choose one
Example: check if Q is V-S:
\[ Q = r_1(A) w_2(A) w_1(A) w_3(A) \]
\[ Q' = w_b(A) r_1(A) w_2(A) w_1(A) w_3(A) \]

\[
\overset{\text{rule 3(a)}}{T_3} \quad \overset{\text{rule 3(b)}}{T_f}
\]

LP(S) acyclic!!
S is V-S

Another example:
\[ Z = w_b(A) r_1(A) w_2(A) r_3(A) w_1(A) w_3(A) \]

\[
\overset{\text{LP(Z) acyclic, so Z is V-S}}{T_f} \quad \overset{\text{equiv. to T_1 T_2 T_3 T_f}}{T_1} \quad \overset{\text{pick one of "1" pair}}{T_3}
\]

S_2 = w_b(A) r_1(A) w_1(A) w_2(A) r_3(A) w_3(A) \n(\text{equivalent to T_1 T_2 T_3 T_f})

Example on useless transactions:
\[ S = w_1(A) r_2(A) w_2(B) r_3(A) w_3(A) w_3(B) \]
\[ S' = \]
\[ T_b \quad w_1(A) r_2(A) w_2(B) r_1(B) w_3(A) w_3(B) \]
\[ \overset{\text{T_b}}{T_1} \quad \overset{\text{T_3}}{T_f}
\]
• If we only care about final state
  – remove T₁, T₂
  - i.e., remove useless transactions
• If we care what T₁, T₂ read (view equivalence), then do not remove useless transactions