CS 277: Database System Implementation

Notes 11: View Serializability

Arthur Keller
View Serializability

Conflict equivalent | View equivalent
Conflict serializable | View serializable
Motivating example

Schedule Q

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

\[ Q = r_1(A) \ w_2(A) \ w_1(A) \ w_3(A) \]

\[
\begin{align*}
P(Q): \quad & T_1 \leftrightarrow T_2 \\
& \downarrow \\
& T_3
\end{align*}
\]

\[ \} \quad \text{Not conflict serializable!} \]
But now compare Q to Ss, a serial schedule:

Q    T₁    T₂    T₃
       Read(A)
       Write(A)

Ss    T₁    T₂    T₃
      Read(A)
      Write(A)

Write(A)  Write(A)  Write(A)
• $T_1$ reads same thing in Q, Ss
• $T_2, T_3$ read something (nothing?)
• After Q or Ss, DB is left in same state

\ So what is wrong with Q?
Definition  Schedules $S_1, S_2$ are View Equivalent if:

1. If in $S_1$: $w_j(A) \Rightarrow ri(A)$
   then in $S_2$: $w_j(A) \Rightarrow ri(A)$

2. If in $S_1$: $ri(A)$ reads initial DB value,
   then in $S_2$: $ri(A)$ also reads initial DB value

3. If in $S_1$: $T_i$ does last write on $A$,
   then in $S_2$: $T_i$ also does last write on $A$

⇒ means “reads value produced”
Definition

Schedule $S_1$ is **View Serializable** if it is view equivalent to some serial schedule.
View Serializable $\iff$ Conflict Serializable

- View Serializable $\not\implies$ Conflict Serializable
  
e.g., See Schedule Q

- Conflict Serializable $\implies$ View Serializable
Lemma
Conflict Serializable $\Rightarrow$ View Serializable

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

All schedules

View Serializable

Conflict

Serializable
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?

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 this write exists
• Another problem: useless writes

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

\[ \text{no } A \text{ reads} \]
To check if S is View Serializable

(1) Add final transaction $T_f$ that reads all DB
   (eliminates condition 3 of V-S definition)

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

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

E.g.: $S = \text{wb}(A) \ldots \text{r1}(A) \ldots \text{w2}(A) \ldots$

add
(3) Create labeled precedence graph of $S$:
(3a) If $w_i(A) \Rightarrow r_j(A)$ in $S$, add $T_i \rightarrow T_j$
(3b) For each \( w_i(A) \Rightarrow r_j(A) \) do

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

- If \( T_i \neq T_b \land T_j \neq T_f \) then insert

\[
\begin{align*}
T_k \xrightarrow{p} T_i \\
T_j \xrightarrow{p} T_k
\end{align*}
\]

some new \( p \)

- If \( T_i = T_b \land T_j \neq T_f \) then insert

\[
T_j \xrightarrow{0} T_k
\]

- If \( T_i \neq T_b \land T_j = T_f \) then insert

\[
T_k \xrightarrow{0} T_i
\]
(4) Check if LP(S) is “acyclic” (if so, S is V-S)
   - For each pair of “p” arcs (p ≠ 0), choose one
Example: check if $Q$ is V-S:

$Q = r_1(A) \cdot w_2(A) \cdot w_1(A) \cdot w_3(A)$

$Q' = w_b(A) \Rightarrow r_1(A) \cdot w_2(A) \cdot w_1(A) \cdot w_3(A) \Rightarrow r_f(A)$

LP(S) acyclic!!

$S$ is V-S
Another example:
\[ Z = w_b(A) \Rightarrow r_1(A) \; w_2(A) \Rightarrow r_3(A) \; w_1(A) \; w_3(A) \Rightarrow r_f(A) \]

LP(Z) acyclic, so Z is V-S (equivalent to \( T_b \; T_1 \; T_2 \; T_3 \; T_f \))

do not pick this one of “1” pair
\[ S_s = w_b(A)r_1(A)w_1(A)r_2(A)w_2(A)r_3(A)w_3(A)r_f(A) \]

\[ T_1 \quad T_2 \quad T_3 \]

\[ Z + S_s \text{ indeed do same thing} \]
• Checking view serializability is expensive
• Still, V-S useful in some cases...
Example on useless transactions:

\[ S = w_1(A) \; r_2(A) \; w_2(B) \; r_1(B) \; w_3(A) \; w_3(B) \]
\[ S' = \]
\[ T_b \ w_1(A) \Rightarrow r_2(A) w_2(B) \Rightarrow r_1(B) \ w_3(A) w_3(B) \Rightarrow T_f \]

\[
\begin{array}{c}
\text{T}_b \\
\text{T}_2 \\
\text{T}_1 \\
\text{T}_3 \rightarrow 0 \\
\end{array}
\]
• If we only care about final state \}
  remove \(T_1, T_2\); i.e., remove useless transactions

• If we care what \(T_1, T_2\) read (view equivalence), then do not remove useless transactions
• If all transactions read what they write, (I.e., $T_j = \ldots R_j(A) \ldots W_j(A)\ldots$) then view serializability = conf. serializability

[Another way of saying: blind writes appear in any view-serializable schedule that is not conflict serializable]
Proof(?): say $S_1$ is view-ser. and no blind writes. $S_1$ V-equiv to $S_s$, serial schedule.

1) Goal: Show that
   $T_1 \rightarrow T_2$ in $P(S_1) \Rightarrow T_1 <_{ss} T_2$

2) Assume $T_1 \rightarrow T_2$
   
   if $S_1 = \ldots w_1(A) \ldots r_2(A)\ldots$
   (direct read) clearly $T_1 <_{ss} T_2$

   if $S_1 = \ldots w_1(A)\ldots r_3(A) w_3(A) \ldots r_2(A)\ldots$
   also $T_1 <_{ss} T_2$

   if $S_1 = \ldots r_1(A) r_3(A) \ldots w_1(A) \ldots w_3(A) \ldots r_2(A)$
   not possible: $T_1, T_3$ not serializable

Other cases similar...
Implications:

If no blind writes, view-ser ⇐⇒ conf-ser

P(S) acyclic ⇒ all transactions read the same as in a serial schedule