Schedules and Concurrency

- Want schedules that are “good”, regardless of
  - initial state and
  - transaction semantics
- Only look at order of read and writes

- Example:
  - Sc=r1(A)w1(A)r2(A)w2(A)r1(B)w1(B)r2(B)w2(B)
Example

$$Sc = r_1(A)w_1(A)r_2(A)w_2(A)r_1(B)w_1(B)r_2(B)w_2(B)$$

$$Sc' = r_1(A)w_1(A)r_1(B)w_1(B)r_2(A)w_2(A)r_2(B)w_2(B)$$

$$T_1 \quad T_2$$

3/26/2021

However, for $Sd$:

$$Sd = r_1(A)w_1(A)r_2(A)w_2(A)r_2(B)w_2(B)r_1(B)w_1(B)$$

- as a matter of fact,
  $T_2$ must precede $T_1$
  in any equivalent schedule,
  i.e., $T_2 \rightarrow T_1$
• $T_2 \rightarrow T_1$

• Also, $T_1 \rightarrow T_2$

$T_1 \xrightarrow{\text{Sd cannot be rearranged}} T_2$  
$\Rightarrow$  
$\text{Sd is not “equivalent” to}$  
$\Rightarrow$  
$\text{any serial schedule}$  
$\Rightarrow$  
$\text{Sd is “bad”}$

Returning to Sc

$Sc = r_1(A)w_1(A)r_2(A)w_2(A)r_1(B)w_1(B)r_2(B)w_2(B)$

$T_1 \rightarrow T_2$  
$T_1 \rightarrow T_2$

$\blacklozenge \text{ no cycles } \Rightarrow \text{Sc is “equivalent” to a serial schedule}$  
(in this case $T_1,T_2$)
**Concepts**

**Transaction:** sequence of $r_i(x), w_i(x)$ actions

**Conflicting actions:**

$$ r_1(A) \leftarrow w_2(A) \leftarrow w_1(A) \leftarrow w_2(A) \leftarrow r_1(A) $$

**Schedule:** represents chronological order in which actions are executed

**Serial schedule:** no interleaving of actions or transactions

---

**What about concurrent actions?**

- Ti issues read($x,t$)
- System issues input($x$)
- Input($X$) completes
- $t \leftarrow x$

- T2 issues write($B,S$)
- System issues input($B$)
- input($B$) completes
- System issues output($B$)
- B $\leftarrow S$

- output($B$) completes
• So net effect is either
  – \( S = \ldots r_1(x) \ldots w_2(b) \ldots \) or
  – \( S = \ldots w_2(B) \ldots r_1(x) \ldots \)

What about conflicting, concurrent actions on same object?

\[
\begin{array}{c}
\text{start } r_1(A) \\
\text{+} \\
\text{start } w_2(A)
\end{array}
\quad
\begin{array}{c}
\text{end } r_1(A) \\
\text{+} \\
\text{end } w_2(A)
\end{array}
\quad
\text{time}
\]

• Assume equivalent to either \( r_1(A) \ w_2(A) \)
  or \( w_2(A) \ r_1(A) \)

• \( \Rightarrow \) low level synchronization mechanism
• Assumption called “atomic actions”
Serializability

- **Basic Assumption** – Each transaction preserves database consistency.
- Thus, serial execution of a set of transactions preserves database consistency.
- A (possibly concurrent) schedule is serializable if it is equivalent to a serial schedule. Different forms of schedule equivalence give rise to the notions of:
  1. **Conflict serializability**
  2. **View serializability**

Simplifying assumptions
- We ignore operations other than *read* and *write* instructions
- We assume that transactions may perform arbitrary computations on data in local buffers in between reads and writes.
- Our simplified schedules consist of only *read* and *write* instructions

Conflicting Instructions

Instructions \( l_i \) and \( l_j \) of transactions \( T_i \) and \( T_j \) respectively, conflict if and only if there exists some item \( Q \) accessed by both \( l_i \) and \( l_j \), and at least one of these instructions wrote \( Q \).

1. \( l_i = \text{read}(Q) \), \( l_j = \text{read}(Q) \). \( l_i \) and \( l_j \) don’t conflict.
2. \( l_i = \text{read}(Q) \), \( l_j = \text{write}(Q) \). They conflict.
3. \( l_i = \text{write}(Q) \), \( l_j = \text{read}(Q) \). They conflict
4. \( l_i = \text{write}(Q) \), \( l_j = \text{write}(Q) \). They conflict

Intuitively, a conflict between \( l_i \) and \( l_j \) forces a (logical) temporal order between them.

If \( l_i \) and \( l_j \) are consecutive in a schedule and they do not conflict, their results would remain the same even if they had been interchanged in the schedule.
Definition

- S1, S2 are conflict equivalent schedules
  - if S1 can be transformed into S2 by a series of swaps on non-conflicting actions.
- A schedule is *conflict serializable* if it is conflict equivalent to some serial schedule.

Conflict Serializability (Cont.)

- Schedule 3 can be transformed into Schedule 6, a serial schedule where $T_2$ follows $T_1$, by series of swaps of non-conflicting instructions. Therefore Schedule 3 is conflict serializable.

<table>
<thead>
<tr>
<th>$T_1$</th>
<th>$T_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>read (A)</td>
<td>read (A)</td>
</tr>
<tr>
<td>write (A)</td>
<td>write (A)</td>
</tr>
<tr>
<td>read (B)</td>
<td>read (B)</td>
</tr>
<tr>
<td>write (B)</td>
<td>write (B)</td>
</tr>
</tbody>
</table>

Schedule 3

<table>
<thead>
<tr>
<th>$T_1$</th>
<th>$T_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>read (A)</td>
<td>read (A)</td>
</tr>
<tr>
<td>write (A)</td>
<td>write (A)</td>
</tr>
<tr>
<td>read (B)</td>
<td>read (B)</td>
</tr>
<tr>
<td>write (B)</td>
<td>write (B)</td>
</tr>
</tbody>
</table>

Schedule 6
Conflict Serializability (Cont.)

- Example of a schedule that is not conflict serializable:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_3$</td>
<td>$T_4$</td>
</tr>
<tr>
<td>read (Q)</td>
<td>write (Q)</td>
</tr>
<tr>
<td>write (Q)</td>
<td></td>
</tr>
</tbody>
</table>

- We are unable to swap instructions in the above schedule to obtain either the serial schedule $< T_3, T_4 >$, or the serial schedule $< T_4, T_3 >$.

View Serializability

- Let $S$ and $S'$ be two schedules with the same set of transactions. $S$ and $S'$ are view equivalent if the following three conditions are met, for each data item $Q$,
  1. If in schedule $S$, transaction $T_i$ reads the initial value of $Q$, then in schedule $S'$ also transaction $T_i$ must read the initial value of $Q$.
  2. If in schedule $S$ transaction $T_i$ executes read($Q$), and that value was produced by transaction $T_j$ (if any), then in schedule $S'$ also transaction $T_i$ must read the value of $Q$ that was produced by the same write($Q$) operation of transaction $T_j$.
  3. The transaction (if any) that performs the final write($Q$) operation in schedule $S$ must also perform the final write($Q$) operation in schedule $S'$.

- As can be seen, view equivalence is also based purely on reads and writes alone.
View Serializability (Cont.)

- A schedule $S$ is **view serializable** if it is view equivalent to a serial schedule.
- Every conflict serializable schedule is also view serializable.
- Below is a schedule which is view-serializable but *not* conflict serializable.

<table>
<thead>
<tr>
<th>$T_{27}$</th>
<th>$T_{28}$</th>
<th>$T_{29}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>read ($Q$)</td>
<td>write ($Q$)</td>
<td>write ($Q$)</td>
</tr>
<tr>
<td>write ($Q$)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- What serial schedule is above equivalent to?
- Every view serializable schedule that is not conflict serializable has **blind writes**.

Other Notions of Serializability

- The schedule below produces same outcome as the serial schedule $< T_1, T_5 >$, yet is not conflict equivalent or view equivalent to it.

<table>
<thead>
<tr>
<th>$T_1$</th>
<th>$T_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>read ($A$) $\quad A := A - 50$ write ($A$)</td>
<td>read ($B$) $\quad B := B - 10$ write ($B$)</td>
</tr>
<tr>
<td>read ($B$) $\quad B := B + 50$ write ($B$)</td>
<td></td>
</tr>
</tbody>
</table>

- Determining such equivalence requires analysis of operations other than read and write.
Testing for Serializability

- Consider some schedule of a set of transactions \( T_1, T_2, \ldots, T_n \)
- **Precedence graph** — a direct graph where the vertices are the transactions (names).
- We draw an arc from \( T_i \) to \( T_j \) if the two transactions conflict, and \( T_i \) accessed the data item on which the conflict arose earlier.
- We may label the arc by the item that was accessed.
- Example of a precedence graph

![Graph with transactions T1 and T2]

---

**Exercise:**

- What is \( P(S) \) for
  \[ S = w_3(A) \, w_2(C) \, r_1(A) \, w_1(B) \, r_1(C) \, w_2(A) \, r_4(A) \, w_4(D) \]

- Is \( S \) serializable?