Data-Intensive Computing Systems

Concurrency Control

Shivnath Babu
Transaction

- Programming abstraction
- Implement real-world transactions
  - Banking transaction
  - Airline reservation
Transaction: Programmer’s Role

Consistent State \(\rightarrow\) Transaction \(\rightarrow\) Consistent State
Transaction: System’s Role

• Atomicity
  – All changes of the transaction recorded or none at all

• Durability
  – All future transactions see the changes made by this transaction if it completes

• Isolation
  – Net effect as if the transaction executed in isolation
Transaction: States

Begin → Run → Abort
         ↓                 ↓
          ↓             ↓
         Commit
Transactions

• Historical note:
  – Turing Award for Transaction concept

• Interesting reading:

  Transaction Concept: Virtues and Limitations
  by Jim Gray

Context

- We have seen:
  - Ensure atomicity in presence of failures
- Next:
  - Ensure Isolation during concurrency
Issues with Concurrency: Example

Bank database: 3 Accounts

Account Balances

- A = 500
- B = 500
- C = 500

Property: A + B + C = 1500

Money does not leave the system
Issues with Concurrency: Example

Transaction T1: Transfer 100 from A to B

A = 500, B = 500, C = 500

Read (A, t)

$t = t - 100$

Write (A, t)

Read (B, t)

$t = t + 100$

Write (B, t)

A = 400, B = 600, C = 500
Issues with Concurrency: Example

Transaction T2: Transfer 100 from A to C

Read (A, s)  
s = s - 100  
Write (A, s)  
Read (C, s)  
s = s + 100  
Write (C, s)
<table>
<thead>
<tr>
<th>Transaction T1</th>
<th>Transaction T2</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read (A, t)</td>
<td>Read (A, s)</td>
<td>500</td>
<td>500</td>
<td>500</td>
</tr>
<tr>
<td>t = t - 100</td>
<td>s = s - 100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write (A, t)</td>
<td>Write (A, s)</td>
<td>400</td>
<td>500</td>
<td>500</td>
</tr>
<tr>
<td>Read (B, t)</td>
<td>Read (C, s)</td>
<td>400</td>
<td>600</td>
<td>500</td>
</tr>
<tr>
<td>t = t + 100</td>
<td>s = s + 100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write (B, t)</td>
<td>Write (C, s)</td>
<td>400</td>
<td>600</td>
<td>600</td>
</tr>
</tbody>
</table>

400 + 600 + 600 = 1600
<table>
<thead>
<tr>
<th>Transaction T1</th>
<th>Transaction T2</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read (A, t)</td>
<td>Read (A, s)</td>
<td>500</td>
<td>500</td>
<td>500</td>
</tr>
<tr>
<td>( t = t - 100 )</td>
<td>( s = s - 100 )</td>
<td>400</td>
<td>500</td>
<td>500</td>
</tr>
<tr>
<td>Write (A, t)</td>
<td>Write (A, s)</td>
<td>300</td>
<td>500</td>
<td>500</td>
</tr>
<tr>
<td>Read (B, t)</td>
<td>Read (C, s)</td>
<td>300</td>
<td>600</td>
<td>500</td>
</tr>
<tr>
<td>( t = t + 100 )</td>
<td>( s = s + 100 )</td>
<td>300</td>
<td>600</td>
<td>600</td>
</tr>
<tr>
<td>Write (B, t)</td>
<td>Write (C, s)</td>
<td>300</td>
<td>600</td>
<td>600</td>
</tr>
</tbody>
</table>

\[300 + 600 + 600 = 1500\]
Terminology

• Schedule:
  – The exact sequence of (relevant) actions of one or more transactions
Problems

• Which schedules are “correct”?  
  – Mathematical characterization

• How to build a system that allows only “correct” schedules?  
  – Efficient procedure to enforce correctness
Correct Schedules: Serializability

• Initial database state is consistent
• Transaction:
  – consistent state $\rightarrow$ consistent state
• Serial execution of transactions:
  – Initial state $\rightarrow$ consistent state
• Serializable schedule:
  – A schedule equivalent to a serial schedule
  – Always “correct”
### Serial Schedule

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>Write (A, t)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Read (A, t)</td>
<td></td>
</tr>
<tr>
<td>t</td>
<td>= t - 100</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Write (A, t)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Read (B, t)</td>
<td></td>
</tr>
<tr>
<td>t</td>
<td>= t + 100</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Write (B, t)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Read (C, s)</td>
<td></td>
</tr>
<tr>
<td>s</td>
<td>= s - 100</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Write (A, s)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Read (C, s)</td>
<td></td>
</tr>
<tr>
<td>s</td>
<td>= s + 100</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Write (C, s)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>500</td>
<td>500</td>
<td>500</td>
</tr>
<tr>
<td>T2</td>
<td>400</td>
<td>600</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>300</td>
<td>600</td>
<td>600</td>
</tr>
</tbody>
</table>

\[300 + 600 + 600 = 1500\]
Serial Schedule

T2

Read (A, s)
s = s - 100
Write (A, s)
Read (C, s)
s = s + 100
Write (C, s)

T1

Read (A, t)
t = t - 100
Write (A, t)
Read (B, t)
t = t + 100
Write (B, t)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>500</td>
<td>500</td>
</tr>
</tbody>
</table>

300 + 600 + 600 = 1500
Serial Schedule

Consistent States

S0 → S1 → S2 → ... → Sn

T1 → T2 → Tn
Is this Serializable?

Transaction T1

Read (A, t)
t = t - 100
Write (A, t)

Read (B, t)
t = t + 100
Write (B, t)

Transaction T2

Read (A, s)
s = s - 100
Write (A, s)

Read (C, s)
s = s + 100
Write (C, s)
<table>
<thead>
<tr>
<th>Transaction T1</th>
<th>Transaction T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read (A, t)</td>
<td>Read (A, s)</td>
</tr>
<tr>
<td>Write (A, t)</td>
<td>s = s - 100</td>
</tr>
<tr>
<td>Read (B, t)</td>
<td>Write (A, s)</td>
</tr>
<tr>
<td>t = t - 100</td>
<td>Read (C, s)</td>
</tr>
<tr>
<td>Write (B, t)</td>
<td>s = s + 100</td>
</tr>
<tr>
<td>t = t + 100</td>
<td>Write (C, s)</td>
</tr>
</tbody>
</table>
Is this Serializable?

Transaction T1

Read (A, t)
t = t - 100
Write (A, t)
Read (B, t)
t = t + 100
Write (B, t)

Transaction T2

Read (A, s)
s = s - 100
Write (A, s)
Read (C, s)
s = s + 100
Write (C, s)

No. In fact, it leads to inconsistent state
Is this Serializable?

Transaction T1

Read (A, t)
\[ t = t - 100 \]
Write (A, t)

Read (B, t)
\[ t = t + 100 \]
Write (B, t)

Transaction T2

Read (A, s)
\[ s = s - 100 \]
Write (A, s)

Read (C, s)
\[ s = s + 100 \]
Write (C, s)
Is this Serializable?

Transaction T1

Read (A, t)
\[ t = t - 100 \]
Write (A, t)

Read (B, t)
\[ t = t + 100 \]
Write (B, t)

Transaction T2

Read (A, s)
\[ s = s - 0 \]
Write (A, s)

Read (C, s)
\[ s = s + 0 \]
Write (C, s)

Yes, T2 is no-op
Serializable Schedule

Transaction T1
Read (A, t)
\[ t = t - 100 \]
Write (A, t)
Read (B, t)
\[ t = t + 100 \]
Write (B, t)

Transaction T2
Read (A, s)
\[ s = s - 0 \]
Write (A, s)
Read (C, s)
\[ s = s + 0 \]
Write (C, s)

Serializability depends on code details
 Serializable Schedule

Transaction T1

Read (A, t)
Write (A, t)

Read (B, t)
Write (B, t)

Transaction T2

Read (A, s)
Write (A, s)

Read (C, s)
Write (C, s)

Still Serializable!
Serializability

• General Serializability:
  – Hard to determine

• Goal: weaker serializability
  – Determined from database operations alone

• Database Operations:
  – Reads, Writes, Inserts, …
Simpler Notation

$r_T(X)$  Transaction T reads X

$w_T(X)$  Transaction T writes X
What is $X$ in $r (X)$?

• $X$ could be any component of a database:
  – Attribute of a tuple
  – Tuple
  – Block in which a tuple resides
  – A relation
  – …
New Notation: Example Schedule

r1(A) w1(A) r2(A) w2(A) r1(B) w1(B) r2(B) w2(B)

---
time
Conflict Serializability

- Weaker notion of serializability
- Depends only on reads and writes
Conflicts Serializability

Serializable Schedules

Conflict
Serializable Schedules
Conflict Serializable Schedule

Transformations: swap non-conflicting actions

S → S1 → S2 → ... → Sn

Conflict Serializable Schedule

Serial Schedule
Transformation: Example

r1(A) w1(A) r2(A) \textcolor{teal}{w2(A)} r1(B) w1(B) r2(B) w2(B)

r1(A) w1(A) r2(A) \textcolor{teal}{r1(B)} \textcolor{teal}{w2(A)} w1(B) r2(B) w2(B)
Non-Conflicting Actions

Two actions are **non-conflicting** if whenever they occur consecutively in a schedule, swapping them does not affect the final state produced by the schedule. Otherwise, they are **conflicting**.
Conflicting or Non-Conflicting?

(Work on paper: Example 1)
Conflicting Actions: General Rules

• Two actions of the same transaction conflict:
  – \( r_1(A) \) \( w_1(B) \)
  – \( r_1(A) \) \( r_1(B) \)

• Two actions over the same database element conflict, if one of them is a write
  – \( r_1(A) \) \( w_2(A) \)
  – \( w_1(A) \) \( w_2(A) \)
Conflict Serializability Examples

(Work on paper: Example 2 and 3)
Testing Conflict Serializability

• Construct precedence graph $G$ for given schedule $S$
• $S$ is conflict-serializable iff $G$ is acyclic
Graph Theory 101

Directed Graph:
Graph Theory 101

Directed Graph:

Edges
Directed Graph:

Cycle
Graph Theory 101

Directed Graph:

Not a cycle
Graph Theory 101

Acyclic Graph: A graph with no cycles
Graph Theory 101

Acyclic Graph:
Testing Conflict Serializability

- Construct precedence graph $G$ for given schedule $S$
- $S$ is conflict-serializable iff $G$ is acyclic
Precedence Graph

• Precedence graph for schedule S:
  – Nodes: Transactions in S
  – Edges: Ti → Tj whenever
    • S: … ri (X) … wj (X) …
    • S: … wi (X) … rj (X) …
    • S: … wi(X) … wj (X) …

Note: not necessarily consecutive
Precedence Graph

• $Ti \rightarrow Tj$ whenever:
  – There is an action of $Ti$ that occurs before a conflicting action of $Tj$. 
Precedence Graph Example

(Work on paper: Example 4)
Testing Conflict Serializability

• Construct precedence graph $G$ for given schedule $S$

• $S$ is conflict-serializable iff $G$ is acyclic
Correctness of precedence graph method

(Work on paper)
Serializability vs. Conflict Serializability

(Work on paper: Example 5)
View Serializability

- A schedule $S$ is view serializable if there exists a serial schedule $S'$, such that the source of all reads in $S$ and $S'$ are the same.
View Serializability Example

View Serializable Schedule

r2(B) w2(A) r1(A) r3(A) w1(B) w2(B) w3(B)

Serial Schedule

r2(B) w2(A) w2(B) r1(A) w1(B) r3(A) w3(B)
View Serializability Example

View Serializable Schedule

r2(B) w2(A) r1(A) r3(A) w1(B) w2(B) w3(B)

Serial Schedule

r2(B) w2(A) w2(B) r1(A) w1(B) r3(A) w3(B)
View Serializability Example

View Serializable Schedule

\[ r_2(B) \, w_2(A) \, r_1(A) \, r_3(A) \, w_1(B) \, w_2(B) \, w_3(B) \]

Serial Schedule

\[ r_2(B) \, w_2(A) \, w_2(B) \, r_1(A) \, w_1(B) \, r_3(A) \, w_3(B) \]
View Serializability Example

View Serializable Schedule

\[ \text{r2}(B) \text{ w2}(A) \text{ r1}(A) \text{ r3}(A) \text{ w1}(B) \text{ w2}(B) \text{ w3}(B) \]

Serial Schedule

\[ \text{r2}(B) \text{ w2}(A) \text{ w2}(B) \text{ r1}(A) \text{ w1}(B) \text{ r3}(A) \text{ w3}(B) \]
View Serializability

Serializable Schedules

Conflict
Serializable Schedules

View Serializable Schedules
Problems

• Which schedules are “correct”?  
  – Serializability theory

• How to build a system that allows only “correct” schedules?  
  – Efficient procedure to enforce correctness serializable schedules
Enforcing Serializability

Strategy: Prevent precedence graph cycles?
Next

• Enforcing serializability
  – Locking-based techniques
  – Timestamp-based techniques
  – Validation-based techniques